UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

Trabalho prático nº1

Servidor de cobertura de Wholesaler

Licenciatura em Engenharia Informática

Sistemas Distribuídos 2022/2023

Trabalho elaborado por:
Bruna Silva al74141
Andre Pereira al74066
Francisco al71506

Data: 16/04/2023

Docentes:
Manuel Cunha
Hugo Paredes
Denis Paulino

Protocolo

O modelo **cliente-servidor** é uma estrutura de aplicação distribuída que distribui as tarefas e cargas de trabalho entre os servidores, que fornecem recursos ou serviços, e os clientes, que fazem o pedido desses recursos.

Normalmente os clientes e servidores comunicam através de uma rede de computadores, em computadores distintos apesar de estes poderem operar no mesmo computador.

Mensagens do cliente para o servidor:

- Mensagem de solicitação de informação: O cliente envia o nome do ficheiro de cobertura que deseja enviar para o servidor onde contém a informação necessária. O ficheiro deve ter o conteúdo no formato: Operadora, Município, Domicílio.
- Mensagem de envio de comando: Contém um identificador de comando e os dados do comando a ser executado pelo servidor.

Mensagens do servidor para o cliente:

- Mensagem de resposta de informação: Contém o resultado da solicitação de informação feita pelo cliente, ou uma mensagem de erro caso a solicitação não possa ser atendida.
- Mensagem de confirmação de comando: Confirma a execução bem-sucedida de um comando solicitado pelo cliente, ou uma mensagem de erro caso o comando não possa ser executado.

Estados do cliente:

- Aguardar resposta do servidor: O cliente enviou uma mensagem de solicitação de informação ou de envio de comando e está a aguardar uma resposta do servidor.
- Aguardar confirmação de comando: O cliente enviou uma mensagem de envio de comando e está a aguardar uma confirmação de que o comando foi executado com sucesso.

Estados do servidor:

- Aguardar mensagem do cliente: O servidor está a aguardar uma mensagem do cliente.
- Processar solicitação: O servidor está a processar uma solicitação de informação ou a execução de um comando enviado pelo cliente.

Neste trabalho iremos usar este modelo para desenvolver um Servidor de cobertura de Wholesaler tendo em conta a localização e interesses do utilizador. Para isso, pretende-se criar um sistema cliente/servidor que implemente um serviço de processamento de ficheiros de cobertura (ficheiros CSV), será implementado um programa cliente e um programa servidor usando a linguagem C# e serão usadas as bibliotecas da framework .NET

(System*). Iremos especificar também um protocolo de comunicação entre o cliente/servidor para o processamento de múltiplos ficheiros.

O serviço de processamento de ficheiros de cobertura deve processar um ficheiro, um CSV enviado pelo cliente, e organizar os domicílios por município. Cada ficheiro enviado pelo cliente pertence a um determinado operador (devidamente identificado) sendo que no caso de cobertura própria se identifica como OWNER. No fim do processamento deverá ser possível indicar, por município, o número de domicílios existentes e, caso existam, as sobreposições existentes em cada domicílio.

O servidor deve receber e responder a vários pedidos simultaneamente usando um modelo de dispachment com threads . Contudo, não deverá processar mais que um ficheiro de determinado operador.
Quando inicialmente contactado, o servidor deve responder com uma mensagem de “100 OK”; Se o servidor receber uma mensagem “QUIT”, deve responder com a mensagem “400 BYE” e terminar a comunicação com o cliente.

O cliente deve receber como parâmetro ou pedir ao utilizador o endereço IP do servidor a contactar. Deve ligar ao servidor e permitir ao utilizador solicitar um ficheiro de cobertura, implementando o protocolo definido e uma interface de texto simples.

Implementação

Nesta parte do relatório iremos apresentar o código executado ao longo do trabalho exemplificando o que foi feito em relação ao atendimento dos clientes, comunicação com cada cliente e o atendimento de múltiplos clientes em simultâneo.

Atendimento dos clientes:

-O servidor utiliza a classe TcpListener para aguardar e aceitar conexões de clientes.
A linha TcpClient client = server.AcceptTcpClient(); aceita a conexão de um cliente.
-Para cada cliente conectado, uma nova thread é criada e iniciada para processar as solicitações do cliente: Thread clientThread = new Thread(new ParameterizedThreadStart(ProcessClient)); clientThread.Start(client); .

```
0 references
public static void Main()
{
    Console.OutputEncoding = Encoding.UTF8;

    CarregarCSV();

    IPAddress ip = IPAddress.Parse("127.0.0.1");
    TcpListener servidor = new TcpListener(ip, 8888);

    servidor.Start();

    Console.WriteLine("Servidor iniciado...");
    while (true)
    {
        TcpClient cliente = servidor.AcceptTcpClient();
        Thread threadCliente = new Thread(new ParameterizedThreadStart(ProcessarCliente));
        threadCliente.Start(cliente);
    }
}
```

Comunicação com cada cliente:
-A comunicação com cada cliente é realizada através da classe NetworkStream e das classes StreamReader e StreamWriter para ler e escrever dados no fluxo de conexão.
-O método ProcessClient é responsável pelo processamento das solicitações do cliente e envio de respostas apropriadas.
-Dentro do método ProcessClient, o servidor lê e escreve dados no fluxo de conexão usando readStream.ReadLine() e writeStream.WriteLine().

```
1 reference
private static void ProcessarCliente(object clienteObj)
{
    TcpClient cliente = (TcpClient)clienteObj;
    NetworkStream fluxoDados = cliente.GetStream();
    StreamReader fluxoLeitura = new StreamReader(fluxoDados, Encoding.UTF8);
    StreamWriter fluxoEscrita = new StreamWriter(fluxoDados, Encoding.UTF8) { AutoFlush = true };

    mutexPrincipal.WaitOne();
    int idClienteAtual = ++clientesConectados;
    mutexPrincipal.ReleaseMutex();

    Console.WriteLine($"Cliente {idClienteAtual} conectado.");

    string empresa = null;
    Mutex mutexEmpresaAtual = null;

    try
    {
        fluxoEscrita.WriteLine("100 OK");

        while (true)
        {
            string comandoRecebido = fluxoLeitura.ReadLine();

            if (comandoRecebido == "QUIT")
            {
                if (mutexEmpresaAtual != null)
                {
                    mutexEmpresaAtual.ReleaseMutex();
                    fluxoEscrita.WriteLine("400 BYE");
                    break;
                }
            }
```

```
else if (comandoRecebido == "SEND")
{
    if (mutexEmpresaAtual != null)
    {
        mutexEmpresaAtual.ReleaseMutex();
    }

    empresa = fluxoLeitura.ReadLine();

    if (!empresas.ContainsKey(empresa))
    {
        mutexPrincipal.WaitOne();
        empresas.Add(empresa, new List<string>());
        mutexEmpresa.Add(empresa, new Mutex());
        mutexPrincipal.ReleaseMutex();
    }
    mutexEmpresaAtual = mutexEmpresa[empresa];
    mutexEmpresaAtual.WaitOne();

    string nomeArquivo = fluxoLeitura.ReadLine();

    List<string> linhasRecebidas = new List<string>();
    string linhaAtual;
    while ((linhaAtual = fluxoLeitura.ReadLine()) != null && linhaAtual != "END")
    {
        linhasRecebidas.Add(linhaAtual);
    }

    linhasRecebidas.RemoveAt(0);

    string caminhoCSV = $"{empresa}.csv";

    if (!File.Exists(caminhoCSV))
    {
        Console.WriteLine($"Arquivo {caminhoCSV} nao encontrado. Criando novo arquivo.");
        File.WriteAllText(caminhoCSV, "Empresa, Localidade, Domicílios, Município\n");
    }
```

```
else
{
    Console.WriteLine($"Arquivo {caminhoCSV} encontrado.");
}
if (empresas.ContainsKey(empresa))
{
    var linhasExistentes = empresas[empresa];
    linhasExistentes.AddRange(linhasRecebidas.Where(linha => !linhasExistentes.Contains(linha)));
    linhasExistentes.Sort((x, y) => x.Split(',')[3].CompareTo(y.Split(',')[3]));
    File.WriteAllLines(caminhoCSV, linhasExistentes, Encoding.UTF8);
}
else
{
    List<string> linhasOrdenadas = new List<string> { "cod_distrito,cod_concelho,cod_localidade,nome_localidade,cod_arteria,tipo_arteria,prep1,titulo_arteria,prep2,nome_arteria,local_arteria,t
    linhasOrdenadas.AddRange(linhasRecebidas);
    linhasOrdenadas.Sort((x, y) => x.Split(',')[3].CompareTo(y.Split(',')[3]));
    File.WriteAllLines(caminhoCSV, linhasOrdenadas, Encoding.UTF8);
    empresas.Add(empresa, linhasOrdenadas);
}
```

Garantia de processamento dos ficheiros de cobertura:
-O servidor mantém um dicionário providers para armazenar os dados de cobertura de cada empresa. Os arquivos CSV são carregados no início do programa no método CarregarCSV().
-Durante o processamento das solicitações dos clientes, o servidor verifica a existência dos arquivos CSV e os atualiza com novos dados conforme necessário.
-As linhas de cobertura enviadas pelos clientes são adicionadas às linhas existentes, e as linhas duplicadas são removidas. O arquivo CSV atualizado é salvo com a chamada File.WriteAllLines().

```
1 reference
private static void CarregarCSV()
{
    string[] arquivos = Directory.GetFiles(".", "*.csv");

    foreach (string arquivo in arquivos)
    {
        string empresa = Path.GetFileNameWithoutExtension(arquivo);
        List<string> linhas = File.ReadAllLines(arquivo, Encoding.UTF8).ToList();
        empresas.Add(empresa, linhas);
        mutexEmpresa.Add(empresa, new Mutex());
    }
}
```

Atendimento simultâneo de múltiplos clientes:
-O servidor utiliza threads para lidar com a conexão de múltiplos clientes simultaneamente.
-Quando um novo cliente se conecta, uma nova thread é criada e iniciada com a chamada clientThread.Start(client).
-O uso de threads permite que o servidor processe as solicitações de diferentes

clientes de forma paralela e independente.
-Os Mutex são usados para garantir a exclusão mútua no acesso a recursos compartilhados, como o dicionário providers e os arquivos CSV de cada operadora. Isso evita condições de corrida e garante a consistência dos dados.

Anexo - Código Fonte

SERVIDOR.CS

```
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;

class Servidor
{
   private static Mutex mutexPrincipal = new Mutex();
   private static Dictionary<string, Mutex> mutexEmpresa = new Dictionary<string, Mutex>();
   private static int clientesConectados = 0;
   private static Dictionary<string, List<string>> empresas = new Dictionary<string, List<string>>();
   private static readonly string arquivoLog = "uploads.log";
   public static void Main()
   {
      Console.OutputEncoding = Encoding.UTF8;

      CarregarCSV();

      IPAddress ip = IPAddress.Parse("127.0.0.1");
      TcpListener servidor = new TcpListener(ip, 8888);

      servidor.Start();

      Console.WriteLine("Servidor iniciado...");
      while (true)
      {
         TcpClient cliente = servidor.AcceptTcpClient();
         Thread threadCliente = new Thread(new ParameterizedThreadStart(ProcessarCliente));
         threadCliente.Start(cliente);
      }
   }

   private static void CarregarCSV()
   {
      string[] arquivos = Directory.GetFiles(".", "*.csv");

      foreach (string arquivo in arquivos)
      {
         string empresa = Path.GetFileNameWithoutExtension(arquivo);
```

```
            List<string> linhas = File.ReadAllLines(arquivo, Encoding.UTF8).ToList();
            empresas.Add(empresa, linhas);
            mutexEmpresa.Add(empresa, new Mutex());
        }
    }

    private static void ProcessarCliente(object clienteObj)
    {
        TcpClient cliente = (TcpClient)clienteObj;
        NetworkStream fluxoDados = cliente.GetStream();
        StreamReader fluxoLeitura = new StreamReader(fluxoDados, Encoding.UTF8);
        StreamWriter fluxoEscrita = new StreamWriter(fluxoDados, Encoding.UTF8) { AutoFlush = true };

        mutexPrincipal.WaitOne();
        int idClienteAtual = ++clientesConectados;
        mutexPrincipal.ReleaseMutex();

        Console.WriteLine($"Cliente {idClienteAtual} conectado.");

        string empresa = null;
        Mutex mutexEmpresaAtual = null;

        try
        {
            fluxoEscrita.WriteLine("100 OK");

            while (true)
            {
                string comandoRecebido = fluxoLeitura.ReadLine();

                if (comandoRecebido == "QUIT")
                {
                    if (mutexEmpresaAtual != null)
                    {
                        mutexEmpresaAtual.ReleaseMutex();
                        fluxoEscrita.WriteLine("400 BYE");
                        break;
                    }
                }
                else if (comandoRecebido == "SEND")
                {
                    if (mutexEmpresaAtual != null)
                    {
                        mutexEmpresaAtual.ReleaseMutex();
                    }

                    empresa = fluxoLeitura.ReadLine();

                    if (!empresas.ContainsKey(empresa))
                    {
                        mutexPrincipal.WaitOne();
                        empresas.Add(empresa, new List<string>());
                        mutexEmpresa.Add(empresa, new Mutex());
                        mutexPrincipal.ReleaseMutex();
                    }
                    mutexEmpresaAtual = mutexEmpresa[empresa];
                    mutexEmpresaAtual.WaitOne();
```

```
            string nomeArquivo = fluxoLeitura.ReadLine();

            List<string> linhasRecebidas = new List<string>();
            string linhaAtual;
            while ((linhaAtual = fluxoLeitura.ReadLine()) != null && linhaAtual != "END")
            {
                linhasRecebidas.Add(linhaAtual);
            }

            linhasRecebidas.RemoveAt(0);

            string caminhoCSV = $"{empresa}.csv";

            if (!File.Exists(caminhoCSV))
            {
                Console.WriteLine($"Arquivo {caminhoCSV} nao encontrado. Criando novo arquivo.");
                File.WriteAllText(caminhoCSV, "Operadora, Localidade, Domicílios, Município\n");
            }
            else
            {
                Console.WriteLine($"Arquivo {caminhoCSV} encontrado.");
            }
            if (empresas.ContainsKey(empresa))
            {
                var linhasExistentes = empresas[empresa];
                linhasExistentes.AddRange(linhasRecebidas.Where(linha => !linhasExistentes.Contains(linha)));
                linhasExistentes.Sort((x, y) => x.Split(',')[3].CompareTo(y.Split(',')[3]));
                File.WriteAllLines(caminhoCSV, linhasExistentes, Encoding.UTF8);
            }
            else
            {
                List<string> linhasOrdenadas = new List<string> {
"cod_distrito,cod_concelho,cod_localidade,nome_localidade,cod_arteria,tipo_arteria,prep1,titulo_arteria,prep2,no
me_arteria,local_arteria,troco,porta,cliente,num_cod_postal,ext_cod_postal,desig_postal" };
                linhasOrdenadas.AddRange(linhasRecebidas);
                linhasOrdenadas.Sort((x, y) => x.Split(',')[3].CompareTo(y.Split(',')[3]));
                File.WriteAllLines(caminhoCSV, linhasOrdenadas, Encoding.UTF8);
                empresas.Add(empresa, linhasOrdenadas);
            }

            RegistrarUpload(empresa, nomeArquivo, idClienteAtual);

            Console.WriteLine($"Arquivo CSV processado para a empresa {empresa}.");

            var localidades = empresas[empresa]
                .Select(linha => linha.Split(','))
                .GroupBy(arr => arr[3])
                .Select(g => new { Localidade = g.Key, Domicílios = g.Count() });

            Console.WriteLine("Resumo das localidades:");
            foreach (var localidade in localidades)
            {
                string saida = $"Localidade: {localidade.Localidade}, Domicilios: {localidade.Domicílios}";
                Console.WriteLine(saida);
                fluxoEscrita.WriteLine(saida);
            }

            int sobreposicoes = ContarSobreposicoes(empresas[empresa], linhasRecebidas);
```

```
                var sobreposicoesPorLocalidade = linhasRecebidas
                    .Select(linha => linha.Split(','))
                    .GroupBy(arr => arr[3])
                    .Select(g => new { Localidade = g.Key, Sobreposicoes = g.Sum(arr =>
ContarSobreposicoes(empresas[empresa], new List<string> { string.Join(",", arr) })) });

                Console.WriteLine("Sobreposiçoes por município:");
                foreach (var sobreposicao in sobreposicoesPorLocalidade)
                {
                    string saida = $"Localidade: {sobreposicao.Localidade}, Sobreposiçoes:
{sobreposicao.Sobreposicoes}";
                    Console.WriteLine(saida);
                    fluxoEscrita.WriteLine(saida);
                }
                fluxoEscrita.WriteLine("400 BYE");
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Erro: {ex.Message}");
    }

    Console.WriteLine($"Cliente {idClienteAtual} desconectado.");
    fluxoDados.Close();
    cliente.Close();
}

private static int ContarSobreposicoes(List<string> linhasServidor, List<string> linhasCliente)
{
    var domiciliosServidor = linhasServidor
        .Select(linha => linha.Split(','))
        .GroupBy(arr => new
        {
            Localidade = arr[3],
            CodArteria = arr[4],
            Porta = arr[12]
        }).ToDictionary(g => g.Key, g => g.Count());
    int sobreposicoes = 0;

    foreach (var linha in linhasCliente)
    {
        var arr = linha.Split(',');
        var chave = new { Localidade = arr[3], CodArteria = arr[4], Porta = arr[12] };

        if (domiciliosServidor.ContainsKey(chave))
        {
            sobreposicoes += domiciliosServidor[chave];
        }
    }

    return sobreposicoes;
}

private static void RegistrarUpload(string empresa, string nomeArquivo, int idCliente)
{
    string dataUpload = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
```

```
        string entradaLog = $"{empresa}, {dataUpload}, {nomeArquivo}, {idCliente}\n";

        if (!File.Exists(arquivoLog))
        {
            File.WriteAllText(arquivoLog, "Operadora, Data Upload, Nome Arquivo, ID Cliente\n", Encoding.UTF8);
        }

        File.AppendAllText(arquivoLog, entradaLog, Encoding.UTF8);
    }
}
```

CLIENTE.CS

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

class ClienteEnvioArquivo
{
    public static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Console.WriteLine("Iniciando conexao");
        using (TcpClient conexao = new TcpClient("127.0.0.1", 8888))
        {
            Console.WriteLine("Conexao estabelecida.");

            using (NetworkStream fluxo = conexao.GetStream())
            {
                StreamReader leitor = new StreamReader(fluxo, Encoding.UTF8);
                StreamWriter escritor = new StreamWriter(fluxo, Encoding.UTF8) { AutoFlush = true };

                string respostaServidor = leitor.ReadLine();
                Console.WriteLine($"Resposta do servidor: {respostaServidor}");

                if (respostaServidor == "100 OK")
                {
                    while (true)
                    {
                        Console.WriteLine("Digite o nome da operadora desejada ou QUIT para encerrar:");
                        string empresa = Console.ReadLine();

                        if (empresa.ToUpper() == "QUIT")
                        {
                            escritor.WriteLine("QUIT");
                            break;
                        }

                        escritor.WriteLine("SEND");
                        escritor.WriteLine(empresa);

                        Console.WriteLine("Informe o caminho do arquivo CSV:");
                        string caminhoArquivo = Console.ReadLine();

                        if (File.Exists(caminhoArquivo))
```

```
                {
                    string[] linhas = File.ReadAllLines(caminhoArquivo, Encoding.UTF8);
                    escritor.WriteLine(Path.GetFileName(caminhoArquivo));

                    Console.WriteLine($"Enviando CSV {Path.GetFileName(caminhoArquivo)}...");
                    foreach (string linha in linhas)
                    {
                        escritor.WriteLine(linha);
                    }
                    escritor.WriteLine("END");
                    Console.WriteLine("CSV enviado. Aguardando processamento...");

                    string informacoesMunicipio;
                    while ((informacoesMunicipio = leitor.ReadLine()) != "400 BYE")
                    {
                        if (informacoesMunicipio == "500 OPERADORA NAO ENCONTRADA")
                        {
                            Console.WriteLine("Operadora nao encontrada.");
                            break;
                        }
                        Console.WriteLine(informacoesMunicipio);
                    }
                }
                else
                {
                    Console.WriteLine("Arquivo CSV nao encontrado.");
                }
            }
        }
    }

    Console.WriteLine("400 BYE");
  }
}
```