Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

REDES DE COMPUTADORES
Segundo Trabalho

Guilherme Marx Ferreira Tavares (14.1.8006)
Gustavo Alves Abreu (14.2.8411)
Professor - Theo Silva Lins

João Monlevade
25 de julho de 2017

Sumário

1 Introdução

A característica do modelo cliente-servidor, descreve a relação de programas numa aplicação. O componente de servidor fornece uma função ou serviço a um ou mais clientes, que iniciam os pedidos de serviço. Tal modelo tornou-se uma das ideias centrais de computação de rede. Muitos aplicativos de negócios, escritos hoje, utilizam o modelo cliente-servidor. O termo também tem sido utilizado para distinguir a computação distribuída por computadores dispersos da "computação"monolítica centralizada em mainframe.

Esta documentação propõe uma abordagem para resolver o seguinte problema proposto: deve ser implementado um servidor Multicast para comunicação de clientes. A comunicação entre processos devem estar situados em sistemas diferentes. O servidor simula o Multicast e deve ser responsável por gerenciar uma rede Multicast, de mesmo ip do servidor. Todas informações e ações dos processos devem ser impressas na tela. "Multicasting é um método ou técnica de transmissão de um pacote de dados para múltiplos destinos ao mesmo tempo. Durante uma transmissão Multicast, o transmissor envia os pacotes de dados somente uma vez."

2 Referencial Teórico

Nesta seção são apresentados alguns termos necessários para uma melhor compreeensão do trabalho desenvolvido.

2.1 Aplicaçoes Cliente Servidor

Cliente-servidor é uma arquitetura de aplicação que estabelece relação entre processos que estão sendo executados em máquinas distintas[Bragança 1999]. Os Hosts fornecedores de um recurso ou serviço são denominados como servidores e os requerentes dos serviços são chamados de clientes.

Geralmente os clientes e servidores comunicam através de uma rede de computadores em computadores distintos, mas tanto o cliente quanto o servidor podem residir no mesmo computador.

2.2 Transmission Control Protocol

O Protocolo de Controle de Transmissão, ou protocolo TCP, é um protocolo da camada de transporte confiável em que existe a garantia que os dados são integralmente transmitidos para os hosts de destino corretos na sequência pelo qual foram enviados. O TCP segmenta a informação proveniente da Camada Aplicação em pequenos blocos de informação, inserindo-lhes um cabeçalho de forma que seja possível no host de destino fazer remontagem dos dados.

Este cabeçalho contém um conjunto de bits (checksum) que permite tanto a validação dos dados como do próprio cabeçalho. A utilização do checksum permite muitas vezes o host de destino recuperar informação em caso de erros simples na transmissão (nos casos da rede corromper o pacote). Caso a informação seja impossível de recuperar ou o pacote TCP/IP se tenha perdido durante a transmissão, é tarefa do TCP voltar a transmitir o pacote. Para que o host de origem tenha a

garantia que o pacote chegou isento de erros, é necessário que o host de destino o informe através do envio de uma mensagem de acknowledgement ou ACK.[Socolofsky and Kale 1991]. Características do protocolo IP:

- Transferência de dados: transmissão ponto-a-ponto de blocos de dados no modo full-duplex ;
- Transferência de dados com diferentes prioridades: transmite em primeiro lugar os datagramas que contenham sinalização de prioridade superior;
- Estabelecimento e liberação de conexões;
- Sequenciação: Ordenação dos pacotes recebidos;
- Segmentação e remontagem: O TCP divide os dados a serem transmitidos em pequenos blocos de dados, identificando-os de forma a que no host de destino seja possível reagrupá-los;
- Controle de fluxo: o TCP é capaz de adaptar a transmissão dos datagramas às condições de transmissões (velocidade , tráfego ...) entre os diversos sistemas envolvidos;
- Controle de erros: A utilização de checksum permite verificar se os dados transmitidos estão livres de erros. É possível, para além da detecção a sua correção;
- Multiplexagem de IP: Uma vez que é utilizado o conceito de portas, é possível enviar dados de diferentes tipos de serviços (portas diferentes) para o mesmo host de destino;

2.3 Thread

Thread pode ser entendida como uma porção mais "leve"de um processo[Tanenbaum 2009]. Como a troca de contexto entre dois processos na CPU pode ser muito cara computacionalmente, o uso de threads possibilita que uma aplicação não fique bloqueada esperando por uma entrada ou recurso, possibilitando a execução de operações paralelamente. A partir disso, surge o conceito de multithreading, que seria um modelo de programação onde o programa pode ser divido em várias threads, podendo ter vários segmentos executados em paralelo.

3 Implementação

Usou-se a linguagem Java para solucionar o problema apresentado devido à experiências anteriores do autor com problemas relacionados usando esta linguagem e também pelo nível de conhecimento do autor sobre a mesma e para isso foi utilizada a IDE Netbeans (8.2) 64 bits.

Para esse problema foi usada uma abordagem em que as aplicações servidoras possuíssem um mecanismo para tratar as várias conexões de aplicações clientes, utilizando o conceito de multithreading.

Para isso, foi desenvolvida uma classe gerenciadora com o estereótipo Handler, que nada mais é do que uma thread que é responsável por tratar cada nova conexão estabelecida com os servidores.

Outro aspecto a se destacar é o uso de troca de mensagens com codificação UTF,que é uma maneira de impedir que aplicações executando em sistemas que usem codificações diferentes, enviem mensagens que fujam do escopo tratado na concepção das mesmas.

Aliado a essas características está o protocolo TCP, que foi escolhido pelas suas características que prezam pela confiabilidade na transmissão de dados.

3.1 Aplicação do Cliente

A aplicação do cliente é composta por duas classes e cinco variáveis:

- Server IP: Uma String que salva o endereço de IP do servidor;
- dis : Variavél de leitura de dados da rede;
- dos : Variavél de escrita de dados na rede;
- requestType: A String que guarda a requisição feita ao servidor;
- In : Scanner responsavel para a leitura de dados do teclado.

Nesta aplicação inicialmente é informado o IP do servidor para que a aplicação se conecte ao mesmo. Caso a conexão seja bem-sucedida então a aplicação cliente exibe um menu na tela com as opções ao cliente.

Este menu possui cinco opções:

- Entrar no grupo: Envia uma RequestType para o servidor solicitando a entrada no grupo e recebe a resposta.
- Sair do grupo: Envia uma RequestType para o servidor solicitando a saida do grupo e recebe a resposta.
- Enviar Mensagem no grupo: Envia uma RequestType para o servidor solicitando o envio de uma mensagem no grupo e já envia a mensagem com a codificação UTF.
- Escutar Mensagens do grupo: Se mantem atento em um loop a todas as mensagens recebidas na variavél dis por quinze segundos.
- Sair do programa: Fecha o socket da conexão e encera o programa.

Após qualquer escolha que não seja sair do programa, o usuário será redicionado para o menu ao fim da execução correspondente.

3.2 Aplicação do Servidor

A aplicação servidor é composta por três classes: HandleConnection, MyServerSocket e MainWindown. A Classe MyServerSocket é a classe principal onde é criado um Socket de servidor para aceitar várias conexões e para cada nova conexão é instanciado um novo objeto da classe gerenciadora Handle Connection para que seja iniciada uma thread para gerenciar esta nova conexão.

Qualquer que seja a exceção lançada, ela necessariamente irá causar o encerramento da aplicação servidora.

A Classe HandleConnection inicializa a entrada e saída de dados, aguardando o RequestType do cliente, que aqui será chamado de ServerAction. Se a requisição for entrada no grupo esta classe irá adicionar este cliente no arraylist de conexões presente na classe MyServerSocket. Se a requisição for de saída esta irá remover este cliente do arraylist. Se a requisição for de enviar mensagem no grupo esta classe irá percorrer o arraylist enviado a cada um dos clientes a mensagem que foi recebida.

A Classe MainWindown é responsável pela interface gráfica.

4 Conclusão

A partir de toda a elaboração deste trabalho, pôde-se perceber o quão é importante um serviço de multicast com a intenção de redirecionar a vários clientes diferentes uma mesma mensagem, além deste serviço ser amplamente utilizado para a navegação na internet.

A maior das dificuldades foi o uso de Threads, isso pois, a sincronização das Threads no programa é muito importante, uma vez que cada cliente só tem uma variável de entrada e uma de saída de dados, então precisou se ter muito cuidado com o uso destas variáveis, para não gerar erros desnecessários.

5 Apêndice A

Código 1: MainClient.java

```
1  /*
2   * To change this license header, choose License Headers in Project
       Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package simpleclient;
7
8  import java.util.Scanner;
9
10 /**
11  *
12  * @author Poxete
13  */
14 public class MainClient {
15     public static void main(String[] args) {
16         // TODO code application logic here
17
```

```
18          Scanner in = new Scanner(System.in);
19
20          System.out.println("Digite o ip do servidor: ");
21          String serverIP = in.nextLine();
22
23          MyClientSocket client = new MyClientSocket(serverIP);
24          client.client();
25      }
26  }
```

Código 2: MyClientSocket.java

```
1   /*
2    * To change this license header, choose License Headers in Project
         Properties.
3    * To change this template file, choose Tools | Templates
4    * and open the template in the editor.
5    */
6   package simpleclient;
7
8   import java.io.DataInputStream;
9   import java.io.DataOutputStream;
10  import java.net.Socket;
11  import java.util.Scanner;
12  /**
13   *
14   * @author Poxete
15   */
16  public class MyClientSocket {
17
18      /**
19       * @param args the command line arguments
20       */
21      private String serverIP = "";
22      private DataInputStream dis = null;
23      private DataOutputStream dos = null;
24      private String requestType = "";
25      static Scanner in = new Scanner(System.in);
26
27      public MyClientSocket(String serverIP) {
28          this.serverIP = serverIP;
29      }
30
31
32
33
34       public void client(){
35           try{
36               Socket connection = new Socket (serverIP,8080);
37               boolean fim = false;
38              do{
39
40
41              dis = new DataInputStream(connection.getInputStream());
42              dos = new DataOutputStream(connection.getOutputStream());
43
44              //aqui define qual o requestType
```

```
45
46              System.out.println("1. Entrar no grupo.");
47              System.out.println("2. Sair do grupo");
48              System.out.println("3. Mandar mensagem no grupo.");
49              System.out.println("4. Escutar grupo por 15 segundos.");
50              System.out.println("5. Sair");
51              System.out.println("Sua opcao: ");
52              int opcao = in.nextInt();
53              in.nextLine();
54              switch(opcao){
55                  case 1:
56                          joinGroup(connection,dis,dos);
57                          break;
58                  case 2: //sai do grupo
59                          leaveGroup(connection,dis,dos);
60                          break;
61                  case 3: sendMessage(connection,dis,dos);
62                          break;
63                  case 4: listenGroup(connection,dis,dos);
64                              break;
65                  case 5: fim = true;
66                          break;
67                  default: System.out.println("Opcao nao valida. Tente
                        novamente.");
68              }
69
70             }while(!fim);
71             if(fim){
72                 dis.close();
73                 dos.close();
74                 connection.close();
75             }
76          }catch(Exception e){
77              e.printStackTrace();
78          }
79      }
80
81          private void joinGroup(Socket connection, DataInputStream dis,
               DataOutputStream dos) {
82              try{
83                  requestType = "Join group";
84
85                  dos.writeUTF(requestType);
86                  dos.flush();
87
88                  String answer = dis.readUTF();
89                  System.out.println(answer);
90
91              }catch(Exception e){
92                  e.printStackTrace();
93              }
94
95
96      }
97
98      private void leaveGroup(Socket connection, DataInputStream dis,
           DataOutputStream dos) {
```

```
99              try{
100                 requestType = "Leave group";
101
102                 dos.writeUTF(requestType);
103                 dos.flush();
104
105                 String answer = dis.readUTF();
106                 System.out.println(answer);
107
108
109             }catch(Exception e){
110                 e.printStackTrace();
111             }
112
113     }
114
115     private void sendMessage(Socket connection, DataInputStream dis,
            DataOutputStream dos) {
116         System.out.println("Digite a mensagem a ser enviada ao grupo: "
                );
117         String msg = in.nextLine();
118             try{
119                 requestType = "Send to group";
120
121                 dos.writeUTF(requestType);
122
123
124                 dos.writeUTF(msg);
125                 dos.flush();
126
127
128
129
130             }catch(Exception e){
131                 e.printStackTrace();
132             }
133
134
135     }
136
137     private void listenGroup(Socket connection, DataInputStream dis,
            DataOutputStream dos) {
138         int i=0;
139         do{
140             try{
141
142                 while (dis.available()>0){
143                     String message = dis.readUTF();
144                     System.out.println(message);
145                 }
146                 Thread.sleep(1000);
147                 i++;
148
149             }catch(Exception e){
150             }
151         }while(i<15);
152     }
```

```
153
154
155 }
```

Código 3: MainWindown.java

```
1
2  public class MainWindown extends javax.swing.JFrame {
3
4      /**
5       * Creates new form MainWindown
6       */
7      public MainWindown() {
8          initComponents();
9      }
10
11     /**
12      * This method is called from within the constructor to initialize
            the form.
13      * WARNING: Do NOT modify this code. The content of this method is
            always
14      * regenerated by the Form Editor.
15      */
16     @SuppressWarnings("unchecked")
17     // <editor-fold defaultstate="collapsed" desc="Generated Code">
18     private void initComponents() {
19
20         connectButton = new javax.swing.JButton();
21         disconnectButton = new javax.swing.JButton();
22         jScrollPane1 = new javax.swing.JScrollPane();
23         infoArea = new javax.swing.JTextArea();
24
25         setDefaultCloseOperation(javax.swing.WindowConstants.
               EXIT_ON_CLOSE);
26
27         connectButton.setText("Connect");
28         connectButton.addActionListener(new java.awt.event.
               ActionListener() {
29             public void actionPerformed(java.awt.event.ActionEvent evt)
                   {
30                 connectButtonActionPerformed(evt);
31             }
32         });
33
34         disconnectButton.setText("Disconnect");
35         disconnectButton.addActionListener(new java.awt.event.
               ActionListener() {
36             public void actionPerformed(java.awt.event.ActionEvent evt)
                   {
37                 disconnectButtonActionPerformed(evt);
38             }
39         });
40
41         infoArea.setColumns(20);
42         infoArea.setRows(5);
43         jScrollPane1.setViewportView(infoArea);
44
```

```
45          javax.swing.GroupLayout layout = new javax.swing.GroupLayout(
                getContentPane());
46          getContentPane().setLayout(layout);
47          layout.setHorizontalGroup(
48                  layout.createParallelGroup(javax.swing.GroupLayout.
                        Alignment.LEADING)
49                      .addGroup(layout.createSequentialGroup()
50                              .addContainerGap()
51                              .addGroup(layout.createParallelGroup(
                                    javax.swing.GroupLayout.Alignment.
                                    TRAILING, false)
52                                      .addComponent(jScrollPane1,
                                            javax.swing.GroupLayout.
                                            Alignment.LEADING)
53                                      .addGroup(javax.swing.
                                            GroupLayout.Alignment.
                                            LEADING, layout.
                                            createSequentialGroup()
54                                              .addComponent(
                                                    connectButton)
55                                              .addGap(18, 18, 18)
56                                              .addComponent(
                                                    disconnectButton)))
57                              .addContainerGap(javax.swing.
                                    GroupLayout.DEFAULT_SIZE, Short.
                                    MAX_VALUE))
58          );
59          layout.setVerticalGroup(
60                  layout.createParallelGroup(javax.swing.GroupLayout.
                        Alignment.LEADING)
61                      .addGroup(layout.createSequentialGroup()
62                              .addContainerGap()
63                              .addGroup(layout.createParallelGroup(
                                    javax.swing.GroupLayout.Alignment.
                                    BASELINE)
64                                      .addComponent(connectButton)
65                                      .addComponent(disconnectButton)
                                            )
66                              .addPreferredGap(javax.swing.
                                    LayoutStyle.ComponentPlacement.
                                    UNRELATED)
67                              .addComponent(jScrollPane1, javax.swing
                                    .GroupLayout.DEFAULT_SIZE, 244,
                                    Short.MAX_VALUE)
68                              .addContainerGap())
69          );
70
71          pack();
72      }// </editor-fold>
73
74      private void connectButtonActionPerformed(java.awt.event.
            ActionEvent evt) {
75          initServerSocket();
76      }
77
78      private void disconnectButtonActionPerformed(java.awt.event.
            ActionEvent evt) {
```

```
79          disconnectServerSocket();
80      }
81
82      /**
83       * @param args the command line arguments
84       */
85      public static void main(String args[]) {
86          java.awt.EventQueue.invokeLater(new Runnable() {
87              public void run() {
88                  new MainWindown().setVisible(true);
89              }
90          });
91      }
92
93      // Variables declaration - do not modify
94      private javax.swing.JButton connectButton;
95      private javax.swing.JButton disconnectButton;
96      private javax.swing.JTextArea infoArea;
97      private javax.swing.JScrollPane jScrollPane1;
98      // End of variables declaration
99
100     private MyServerSocket mySocket;
101
102     public void initServerSocket() {
103         mySocket = new MyServerSocket();
104         Thread thread = new Thread(mySocket);
105         thread.start();
106     }
107
108     public void disconnectServerSocket() {
109         mySocket.disconnect();
110     }
111
112 }
```

Código 4: MyServerSocket.java

```
1
2  import java.net.ServerSocket;
3  import java.net.Socket;
4  import java.net.SocketException;
5  import java.util.ArrayList;
6
7  import javax.swing.JOptionPane;
8
9  public class MyServerSocket implements Runnable {
10
11      private boolean connected = false;
12      private ServerSocket serverSocket = null;
13      private Socket connection = null;
14      private HandleConnection handle = null;
15      public static ArrayList<Socket> group = new ArrayList<>();
16      public MyServerSocket() {
17
18      }
19
20      @Override
```

```
21      public void run() {
22          try {
23              serverSocket = new ServerSocket(8080); //Port 8080
24              JOptionPane.showMessageDialog(null, "Server is running");
25              connected = true;
26              int i=1;
27              while (connected) {
28                  System.out.println("Waiting for connections...");
29                  connection = serverSocket.accept();
30                  handle = new HandleConnection(connection);
31                  Thread handleConnection = new Thread(handle);
32                  handleConnection.start();
33                  System.out.println("Thread to handle connection "+ i +
                        " is running...");
34              }
35          } catch (SocketException e) {
36              System.out.println("Server stopped.");
37              JOptionPane.showMessageDialog(null, "Server stopped.");
38          } catch (Exception e) {
39              e.printStackTrace();
40          }
41      }
42
43      public void disconnect() {
44          try {
45              if (serverSocket != null) {
46                  serverSocket.close();
47                  connected = false;
48              }
49          } catch (Exception e) {
50              e.printStackTrace();
51              JOptionPane.showMessageDialog(null, "An exception occurred.
                    ");
52          }
53      }
54
55  }
```

Código 5: HandleConnection.java

```
1   import java.io.DataInputStream;
2   import java.io.DataOutputStream;
3   import java.io.IOException;
4   import java.net.Socket;
5   import java.util.logging.Level;
6   import java.util.logging.Logger;
7
8   public class HandleConnection implements Runnable {
9
10      private Socket connection;
11      private DataInputStream dis;
12      private DataOutputStream dos;
13      final static String INET_ADDR = "224.0.0.3";
14
15
16      final static int PORT = 8888;
17      public HandleConnection(Socket connection) {
```

```
18          this.connection = connection;
19      }
20
21      @Override
22      public void run() {
23          System.out.println("Handling connection...");
24          while(true){
25          try {
26              dis = new DataInputStream(connection.getInputStream());
27              dos = new DataOutputStream(connection.getOutputStream());
28              String serverAction = dis.readUTF(); //Acao a ser feita,
                    pode ser Send to group, Join group, Leave group
29              String msg2;
30
31              if(serverAction.equals("Join group")){
32                  if(!MyServerSocket.group.contains(connection))
33                          MyServerSocket.group.add(connection);
34
35                  dos.writeUTF("OK");
36                  dos.flush();
37              }else if (serverAction.equals("Leave group")){
38                  MyServerSocket.group.remove(connection);
39                  dos.writeUTF("OK");
40                  dos.flush();
41              }else if (serverAction.equals("Send to group")){
42                  System.out.println("Passou aqui");
43                  msg2 = dis.readUTF();
44                  sendMessage(msg2);
45
46              }
47
48                  /*
49                  if (serverAction.equals("Message to server")) {
50                  receiveMessage(); //Receber mensagem
51                  } else if (serverAction.equals("Message from server"))
                        {
52                  serverAction = dis.readUTF();
53                  sendMessage(serverAction); //Enviar mensagem
54                  }
55                  else if(serverAction.equals("Message from server2")){
56
57                  }*/
58                  //this.wait(1000);
59
60
61
62          } catch (IOException e) {
63              e.printStackTrace();
64          }}
65      }
66
67      private void receiveMessage() throws IOException {
68          String msg = dis.readUTF(); //Le String do cliente
69          System.out.println("Received message: " + msg );
70
71
72      }
```

```
73
74      private void sendMessage(String msg) throws IOException {
75
76          for(Socket auxiliar: MyServerSocket.group){
77              DataOutputStream dos1 = new DataOutputStream(auxiliar.
                    getOutputStream());
78              dos1.writeUTF(msg); //Escreve String para o cliente
79              dos1.flush();
80          }
81          System.out.println("Sent message: " + msg);
82      }
83  }
```

6 Referências

[Bragança 1999] Bragança, A. (1999). Desenvolvimento cliente-servidor. Instituto Superior de Engenharia do Porto.

[Tanenbaum 2009] Tanenbaum, A. S. (2009).Sistemas Operacionais Modernos, volume 1.Prentice Hall, 3ª edition