Relatório Do Projeto De Redes De Computadores, Servidor De Arquivos

Lucas Alves Da Costa.

[1]*LucasAlves16.LA@gmail.com*

[2]Instituto federal de educação, ciência e tecnologia-Goiás
(IFG)

Resumo. *Este Artigo tem como meta Mostrar a funcionalidade do projeto de redes de computadores 'Servidor de Arquivos' Feito na linguagem de programação; Java e IDE NetBeans. Onde citarei os passos de como o servidor se comunica com o cliente, mecanismos utilizados para comunicação e transferência de Arquivos usando o protocolo da camada de transporte (TCP/IP junto ao FTP) e erros e limitações de ambos os lados da aplicação. Objetivo principal: O cliente deve solicitar a lista de arquivos compartilhados pelo servidor e a partir dessa lista escolher qual arquivo deseja baixar e baixa-lo.*

1. Servidor

O servidor é a parte Lógica responsável pelo gerenciamento e envio dos arquivos. Isso é feito através do **TCP/IP** utilizado para prover entrega confiável. Optei pelo *TCP* porque durante a transferência do arquivo é necessário que cada dado enviado chegue na ordem correta e para o destino correto,também é necessário criar um fluxo entre duas maquinas, para isso utilizo o endereço IP e numero de porta. Para facilitar a transferência do arquivo o IP recebe pequenas partes do arquivo e envia ao destino, as classes que trabalham com esses mecanismos estão contidas no pacote; *Java.net**. O servidor quando iniciado fica aguardando que que alguém se conecte a ele através de seu IP e numero de porta, quando alguém se conectar é iniciada uma nova conexão através do método *accept*, a partir dai envio e recebo arquivos em forma de **Streans** que são objetos que permitem que possa enviar e receber qualquer tipo de informação seja ela (*Byte,dado,arquivo ou do teclado...*). Enquanto o servidor Estiver Online ele aceitará conexão com o cliente e enviará a lista de arquivos contidos em sua pasta, não é suportado mais de uma conexão por vez, o TCP não irá negar a conexão ao novo cliente em vez disso irá coloca lo em uma lista de espera até que a porta 5555 seja liberada para o uso, posso finalmente enviar a lista de arquivos compartilhados para o novo cliente.

Como isso é feito?

Isso é feito através dos seguintes passos; foi criada uma função que tem como objetivo gravar os nomes de todos os arquivos contidos em uma pasta padrão em uma variável do tipo *String*, no Java é possível saber o nome de cada arquivo através do método *getName()*.

```
1
2  public static void EnviarLista(Socket Cliente) throws
       IOException  {
3
4        File MeuArquivo;
5       /*Aqui A variavel recebe o nome do diretorio dos arquivos
           */
```

```
6         MeuArquivo = new File("MeuArquivo");
7         File[] Lista;
8         // Aqui a variavel Lista recebe o nome de todos os
             arquivos presentes na pasta "MeuArquivo"
9         Lista = MeuArquivo.listFiles();
10         String Arq2 = " ";
11         int cont = 0;
12        for(File fileTmp: Lista){
13           cont ++;
14           Arq2 += fileTmp.getName()+"\n";
15           /*Aqui a variavel arq2 e incrementada linearmente de
                cada nome de arquivo*/
16
17        }
```

Com a Variável devidamente gravada com os nomes de arquivos ja posso realizar o envio da mesma, a variável é enviada para o cliente através do método *write()* que é bastante usado para enviar objetos do tripo **Stream**, o envio e orientado a mesma conexão estabelecida pelo cliente. Assim que concluído o envio informo na tela o endereço de quem a solicitou, em seguida crio uma função responsável por enviar ao cliente o arquivo solicitado. Isso é feito da seguinte maneira; primeiro aguardo uma mensagem do cliente contendo o nome do arquivo que ele deseja baixar, em seguida pesquiso na pasta o nome do arquivo, caso ele exista será transformado em um aglomerado de Bytes e direcionados a conexão do cliente, após o envio confirmo com uma mensagem na tela informando o endereço de quem baixou e o que baixou e o tamanho do arquivo. Dessa forma posso controlar quem usou meu servidor e oque baixou, em seguida retornamos ao método *accept()* e aguardo que o mesmo ou outro cliente solicite a lista novamente. Essa aplicação tem a limitação de não atender mais de um cliente por vez, isso pode ser resolvido com uma**Tread**, outras limitações são as seguintes; o servidor não tem opção de encerramento e só poderá ser encerrado através de uma interrupção forçada, qualquer exceção como; arquivo não encontrado e se o cliente desconectar durante a execução causarão a abordagem da aplicação.

```
1    public static void UpLoad(Socket Cliente) throws IOException{
2          String NomeArq = "";
3          InputStream Recive;
4          Recive = Cliente.getInputStream();
5          int recebido=0;
6          char data = 0;
7
8   // Recebo o nome do arquivo que o cliente solicitou
9      while ((data = (char)Recive.read()) != 4) {
10          //System.out.print((char)data);
11         recebido=data;
12        NomeArq += data;
13       }
14     OutputStream Tam;
15     Tam = Cliente.getOutputStream();
16
```

```
17              System.out.println("Conex o aceita: " + Cliente.
                   getInetAddress().getHostAddress());
18
19              File myFile = new File ("MeuArquivo//"+NomeArq);// o
                    arquivo e buscado no diretorio e enviado
20              // Lembrando que o arquivo deve esta dentro da pasta
                    "Arquivos".
21              long Tamanho = myFile.length();
22
23              // fragmenta o arquivo em uma cadeia de bytes e
                   envia para o cliente
24                // envia o arquivo (transforma em byte array)
25               byte [] mybytearray  = new byte [(int)myFile.length
                    ()];
26                FileInputStream fis = new FileInputStream(myFile);
27                 BufferedInputStream bis = new BufferedInputStream
                      (fis);
28                  bis.read(mybytearray,0,mybytearray.length);
29                   OutputStream os = Cliente.getOutputStream();
30              System.out.println("Enviando...");// Enquanto o
                   arquivo estiver sendo enviado a msg vai aparecer
31
32               os.write(mybytearray,0,mybytearray.length);
33
34               os.flush();// forca o envio de todos os bytes
35               Cliente.close();// fecha a conexao
36               // Msg Grafica para informar o envio do arquivo
37            JOptionPane.showMessageDialog(null, myFile.getName()," 
                   Aquivo Enviado", 2);
38            // Msg para controlar Downloads feitos por usuarios
39            System.out.println("Cliente: " + Cliente.
                   getInetAddress().getHostAddress() + " Fez Download
                   do aquivo; "+myFile.getName()+" Tamanho: "+Tamanho
                   /1024+" KBs");
40
41
42      }
```

2. Cliente

O lado cliente é um pouco mais completo, possui menu que permite que o usuário permaneça na aplicação e saia quanto quiser, possui também opção de mudança de IP através de uma janela de entrada gráfica. Essa parte trabalha com endereço IP e numero de porta do Servidor, foi desenvolvido baseado nos conceitos de *TCP/IP e FTP*. Funciona da seguinte maneira; após solicitar a lista o cliente escolhe o nome do arquivo incluindo sua extensão e envia de volta para o servidor, então é feito o download do arquivo solicitado na pasta em que a aplicação cliente e executada. O Download é feito da seguinte maneira; a função recebe um aglomerado de Bytes e os grava em ordem correta dentro da pasta da aplicação, os métodos usados foram; *Write()* e o *flush()* para garantir que todos

os Bytes sejam gravados em Arquivo, esse arquivo é salvo com o mesmo nome escolhido para o Download. Assim que finaliza o download é exibida uma janela com uma mensagem de confirmação de download e em seguida é chamada a função menu novamente para que o usuário possa escolher se lista novamente os arquivos, ou se muda o endereço do servidor ou se sai da aplicação. Quando a opção **1** é acionada é listado os arquivos compartilhados mas só é listado se o servidor estiver rodando em sua maquina já que essa opção e configurada com o **IP(127.0.0.1)** que é o *local host* de sua máquina, após a opção **1** digitada será chamada em seguida a função *receberMsg()* que é responsável por receber a lista de arquivos e exibir ao usuário. Em seguida é exibida uma mensagem gráfica que informa como escolher o arquivo, logo após o cliente ter escolhido o arquivo a ser baixado é chamada a função de download. Após retornar ao **menu** caso a opção **2** seja acionada será possível alterar o IP do servidor através de uma janela de entrada gráfica, mas só até retornarmos ao menu novamente, esse novo IP só é válido até a conclusão do download sendo necessário que o usuário digite novamente o novo IP já que o servidor deve rodar em uma máquina padrão. com a opção **3** acionada a aplicação é apenas encerrada através do método *System.exit()* já que não há conexões estabelecidas no momento em que o *menu* é exibido. Alguns problemas dessa aplicação são; Arquivos muito grandes não são baixados porque não é recebido o tamanho do arquivo então o arquivo e baixado encima de um buffer fixo, como a aplicação é de teste o servido padrão da opção 1 do menu e a própria maquina do cliente e se um arquivo não for encontrado no servidor a aplicação é encerrada.

```
1  public static void Menu() throws IOException {
2       // inicia a conexao com o servidor raiz
3          System.out.println("Escolha uma  o p   o :\n [1]. Listar
               Arquivos Compartilhados.\n [2]. Configurar IP do
              Servidor."
4                + "\n [3]. Sair!\n");
5          Scanner Ler;
6          Ler = new Scanner(System.in);
7          int Opc;
8          Socket Cliente;
9            Opc = Ler.nextInt();
10           if(Opc==2){
11
12              String newIP = JOptionPane.showInputDialog("Digite
                    O 'IP' do Servidor!");// Entrada por enterface
                    grafica
13              Cliente = new Socket(newIP,5555);
14              long T = System.currentTimeMillis();
15              System.out.println("Conectado com o novo servidor.
                    . .");
16              long tempoEspera;
17               do{
18                 tempoEspera = System.currentTimeMillis()+
                       100000;
19                     System.out.println("O Servidor esta ocupado!
                           Aguarde......\n");
20
```

```
21                  } while(System.currentTimeMillis()>tempoEspera);
22                   receberMsg(Cliente);
23              }if(Opc==3){
24                  System.exit(0);
25              }
26              else{
27               Cliente = new Socket("127.0.0.1",5555);
28                System.out.println("Conectado com o servidor. . .")
                     ;
29                receberMsg(Cliente);
30              }
31      }
```

3. Considerações finais

Foi mostrado neste relatório o funcionamento das aplicações; cliente e servidor, protocolo utilizado foi o TCP/IP e Parcialmente o FTP que se encontram na classe *java.net*, foi desenvolvido no foco de ser o mais simples e funcional possível... O servidor apenas atende o cliente e informa na tela informações básicas para o controle do usuário, assim que o cliente se conectar ele irá receber a lista de arquivos, o servidor ira aguardar a resposta que no caso é o nome do arquivo, quando receber o nome ele irá retornar a esse cliente o arquivo correspondente a esse nome em forma de Bytes, quando o arquivo for completamente recebido o cliente será informado e logo após retornará ao menu. Os objetivos principais da aplicação foram cumpridos, as demais limitações podem ser resolvidas posteriormente caso necessário.