UNIVERSIDADE FEDERAL DE UBERLÂNDIA FEELT – FACULDADE DE ENGENHARIA ELÉTRICA

PAULO JOSÉ CARMONA TEIXEIRA
11611ECP018

SERVIDOR WEB MULTITHREAD

Processo de desenvolvimento de um servidor web multithread em Java

Uberlândia

2019

PAULO JOSÉ CARMONA TEIXEIRA

SERVIDOR WEB MULTITHREAD

Relatório do curso de Engenharia da Computação referente à disciplina Redes de Computadores, a ser utilizado como método de demonstração e conclusão referentes a implementação de um servidor Web com a permissão de múltiplas requisições.

Professor: Paulo Roberto Guardieiro

Uberlândia

2019

Sumário

1 – Objetivo

Este trabalho tem como objetivo a implementação de um servidor Web MultiThread, ou seja, este servidor é capaz de processar várias requisições de serviços ao mesmo tempo, e executá-las em paralelo.

Implementaremos a versão 1.1 do HTTP, definida na RFC 2616, levando em consideração que a versão 1.1 engloba a versão 1.0, definida na RFC 1945.

O servidor devera atender pedidos de transferências de arquivos gerados através de um browser comum (Internet Explorer, Mozilla Firefox, Google Chrome), que estará disponível através de uma porta especifica (especificada no código), e também deverá ter conexões não persistentes e simultâneas.

Essas especificações foram definidas no roteiro do trabalho.

2 – Introdução

O Servidor Web é responsável por aceitar pedidos em HTTP de clientes, e retorna uma resposta, geralmente resposta esta como paginas web (documentos em HTML), onde esses documentos HTML podem possuir: imagens, documentos, vídeos, e internamente respostas em XML ou até mesmo JSON.

Quando o programa é executado, cria-se um processo servidor, que mantem um *“listening”* que pode ser identificado com um “ouvinte”. Sua função é aguardar as requisições feitas pelo lado do cliente, para assim poder retornar as respostas, usando o protocolo HTTP.

O servidor que implementamos será capaz de mostrar uma pagina HTML simples (onde não usaremos Cascading Style Sheets (CSS) para personalização da pagina web), uma imagem, documentos PDF, será mostrado em diferentes navegadores, e executados alguns testes no mesmo momento, assim, comprovamos que o servidor possui o funcionamento especificado para o Multithreading.

3 – Implementação

Para a implementação do servidor proposto, utilizaremos a linguagem de programação JAVA com as seguintes especificações:

- Java 8.2
- JDK 1.8.211
- JRE 1.8.211

E como IDE usarmos o NetBeans, e com elas obtermos:

- Compilação
- Debug
- Testes

3.1 – Classe *WebServer.java*

A classe main é responsável por estabelecer a conexão, e servir de “ouvinte”, dado que ela possui uma linha especifica que é executada ate que se pare o programa. Além do já citado, é nessa classe que se comunica a porta.

```
package webserver;


/**
 * Servidor Multithread
 *
 * @author Paulo José Carmona Teixeira
 */


/* Imports */
import java.io.*;
import java.net.*;
import java.util.*;
```

```
public class Webserver {

    public static void main(String[] args) throws Exception {

        int porta = 1750;

        ServerSocket socket = new ServerSocket(porta);

        /* Processa os serviços de requisição HTTP num loop infinito */

        while (true) {

            /* Requisição de conexão TCP */
            Socket connection = socket.accept();

            /* Construtor do objeto de requisição de mensagem HTTP */
            RequisicaoHttp requisicao = new RequisicaoHttp(connection);

            /* Cria um novo thread para processar a requisição */
            Thread thread = new Thread(requisicao);

            /* Inicia a Thread, dado que a próxima classe é uma Runnable */
            thread.start();
        }
    }
}
```

3.2 – Classe *RequisicaoHTTP.java*

Classe responsável por executar todo o procedimento, é nela que se cria a conexão com o cliente, e através dessa conexão se processa e executa as requisições feitas pelo cliente.

```
package webserver;


/**
 * Servidor Multithread
 *
 * @author Paulo José Carmona Teixeira
 */


/* Imports */
import java.io.*;
import java.net.*;
import java.util.*;


final class RequisicaoHttp implements Runnable {


    final static String CRLF = "\r\n";
    Socket s;


    /*Construtor*/
    public RequisicaoHttp(Socket s) {
        this.s = s;
    }
```

```
/*Implementação da Interface*/
public void run(){
    try{
        processarRequisicao();
    }

    catch (Exception e){
        System.out.println( "Erro -> " + e.getMessage());
    }
}

private void processarRequisicao() throws IOException {

    /*Criamos o Input e o Output*/
    InputStream is = s.getInputStream();
    DataOutputStream os = new DataOutputStream(s.getOutputStream());

    /*Precisamos de um leitor de Buffer*/
    BufferedReader br = new BufferedReader(new InputStreamReader(is));

    /*Aqui receberemos os dados da requisição HTTP*/
    String requisicao = br.readLine();
```

```
/*Extraimos o nome do Arquivo*/
StringTokenizer tokens = new StringTokenizer(requisicao);
tokens.nextToken();
String nomeArquivo = tokens.nextToken();

/*Arquivo no mesmo diretorio*/
nomeArquivo = "." + nomeArquivo;

/*Agora abrimos o arquivo solicitado*/
FileInputStream fis = null;
boolean arquivoExistente = true;

try {
    fis = new FileInputStream(nomeArquivo);
}

catch (FileNotFoundException fnfe) {
    arquivoExistente = false;
    System.out.println("Erro --> " + fnfe.getMessage());
}

/*Agora criaremos algumas partes para o DEBUG*/
System.out.println("Arquivo em transporte!");
System.out.println("Requisicao --> " + requisicao);
String cabecalho = null;
```

```
while((cabecalho = br.readLine()).length() != 0){
    System.out.println("Cabecalho --> " + cabecalho);
}

/*Agora montaremos a mensagem de resposta*/
String status = null;
String conteudo = null;
String entidade = null;

if(arquivoExistente) {
    status = "HTTP/1.0 200 OK" + CRLF;
    conteudo = "Content-Type: " + tipoDeConteudo(nomeArquivo) + CRLF;
}

else {
    status = "HTTP/1.0 404 Not Found" + CRLF;
    conteudo = "Content-Type: text/html" + CRLF;
    entidade = "<HTML>" + "<HEAD><TITLE>Not Found</TITLE></HEAD>"
    + "<BODY>Not Found</BODY></HTML>";
}

/*Escreveremos os Bytes*/
os.writeBytes(status);
os.writeBytes(conteudo);
os.writeBytes(CRLF);
```

```
    /*Envia os dados*/
    if(arquivoExistente) {
        enviaArquivos(fis, os);
        fis.close();
    }

    else {
        os.writeBytes(entidade);
    }

    /*Finaliza*/
    os.close();
    br.close();
    s.close();
}

/* Aqui fazemos a verificação do tipo de conteúdo, baseado no arquivo */
private String tipoDeConteudo(String nomeArquivo) {
    if (nomeArquivo.endsWith(".htm") || nomeArquivo.endsWith(".html")) {
        return "text/html";
    }

    if (nomeArquivo.endsWith(".ram") || nomeArquivo.endsWith(".ra")) {
        return "audio/x-pn-realaudio";
    }
```

```
        if (nomeArquivo.endsWith(".jpg") || nomeArquivo.endsWith(".jpeg")) {
            return "image/jpeg";
        }

        if (nomeArquivo.endsWith(".pdf")) {
            return "application/pdf";
        }

        return "application/octet-stream";
    }

    private void enviaArquivos(FileInputStream fis, DataOutputStream os) throws
IOException {

    /*Construtor do um buffer para guardar os bytes que irão para o socket */
        byte[] buffer = new byte[1024];
        int bytes = 0;

        /* Copia o arquivo solicitado e envia para o output do socket */
        while ((bytes = fis.read(buffer)) != -1) {
            os.write(buffer, 0, bytes);
        }
    }
}
```

4 – Testes

Como dito anteriormente, segue abaixo os testes aplicados no servidor Web.

4.1 – Requisição Web via navegador Opera

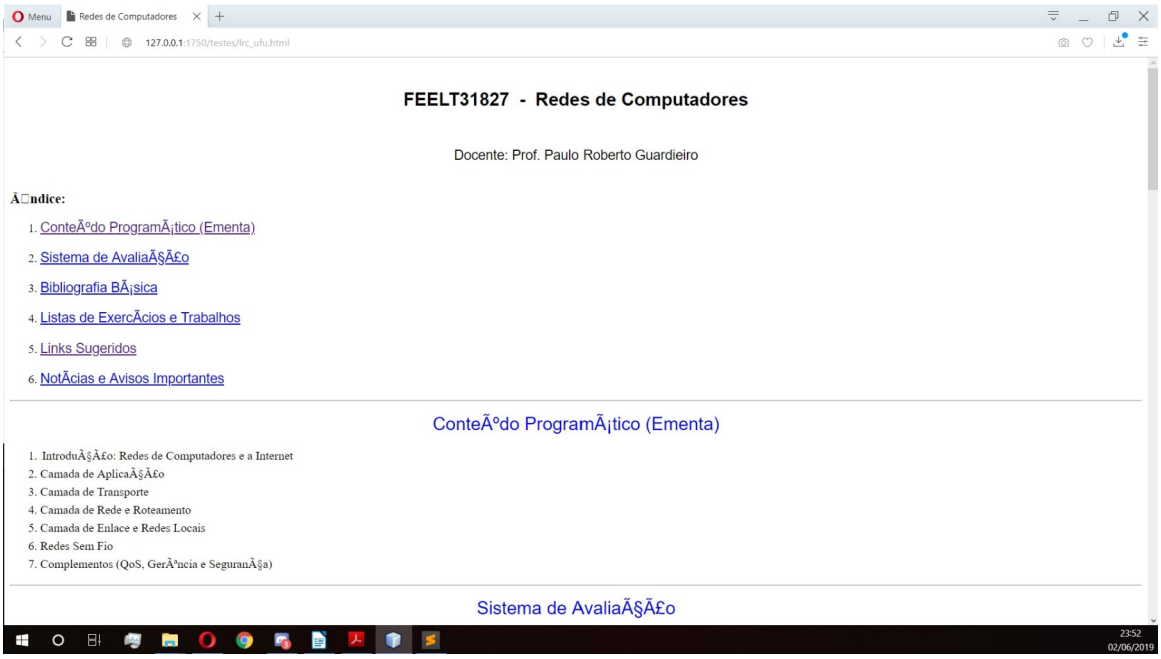

Figura 1 – Imagem da Requisição HTML

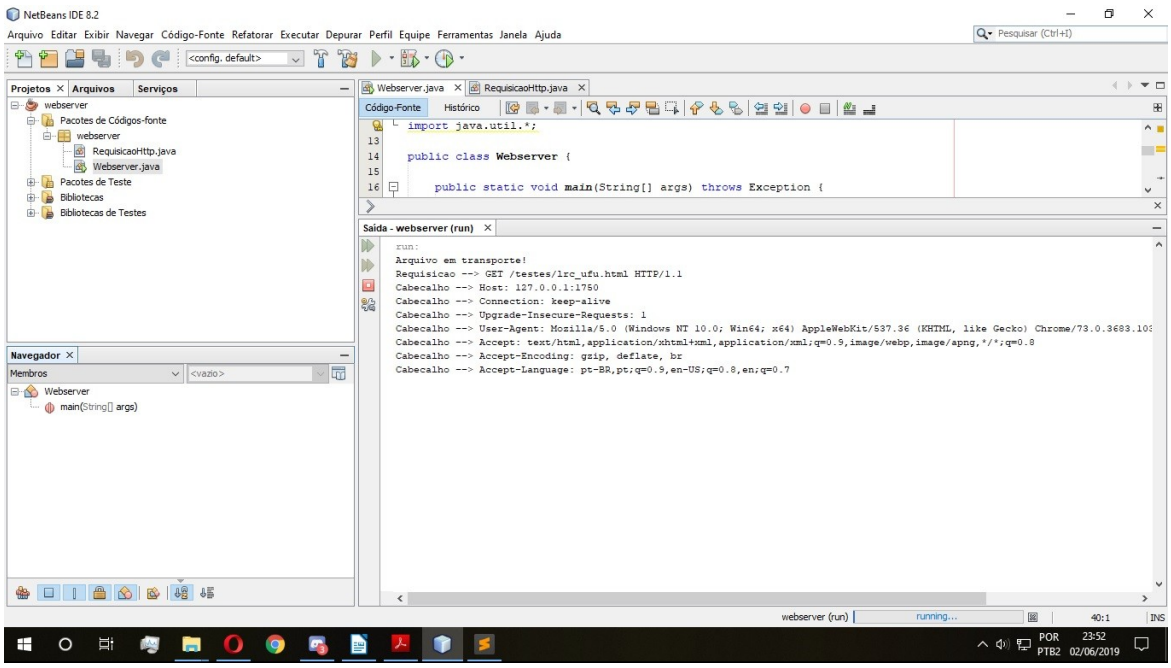

Figura 2 – Imagem das Etapas no Server (NetBeans)

4.2 – Requisição de Imagem via navegador Opera

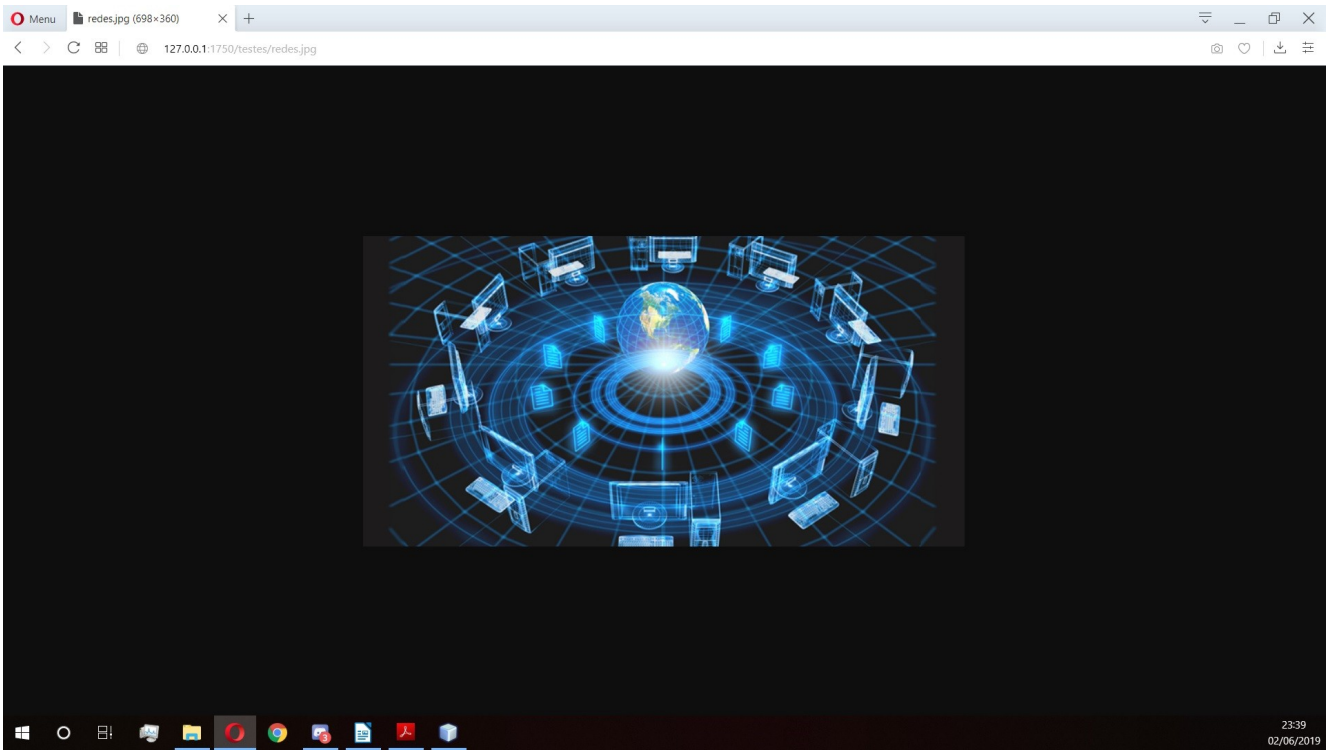

Figura 3 – Imagem da Requisição JPEG

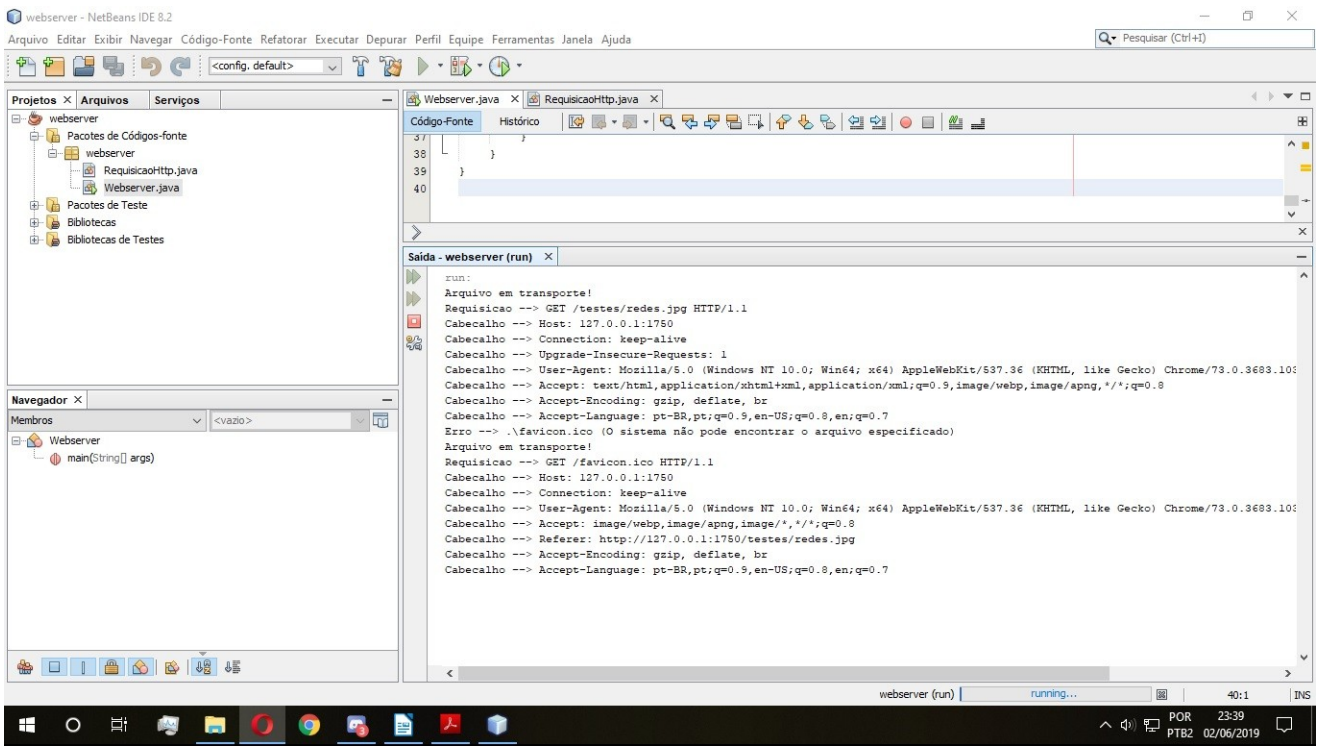

Figura 4 – Imagem das Etapas no Server (NetBeans)

4.3 – Requisição de PDF via navegador Opera

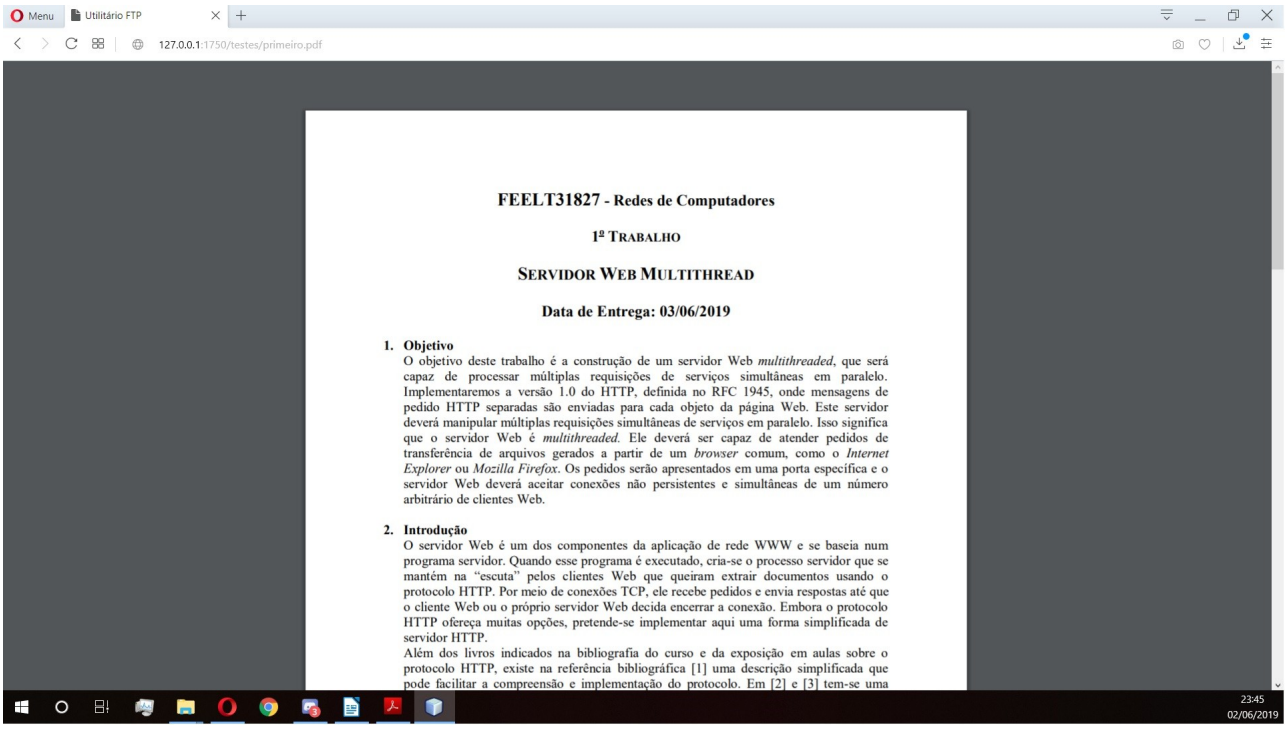

Figura 5 – Imagem da Requisição PDF

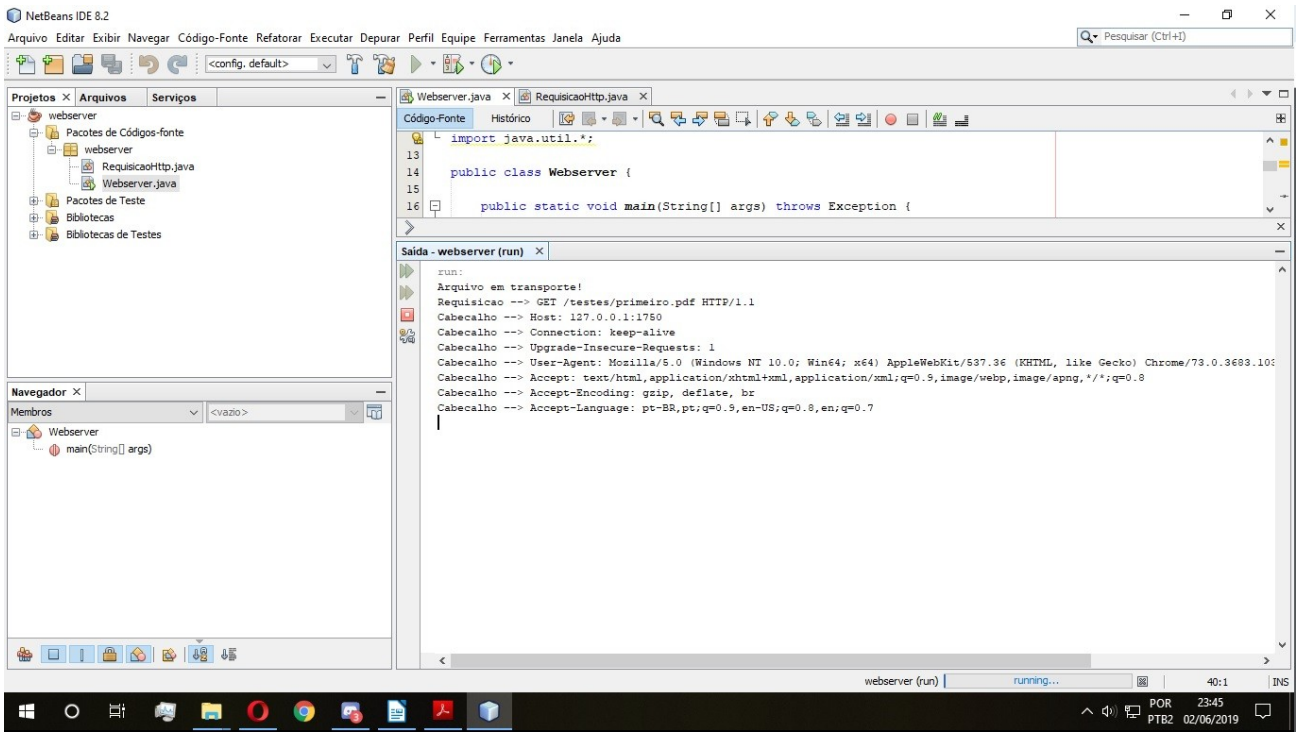

Figura 6 – Imagem das Etapas no Server (NetBeans)

5 – Conclusão

Com este servidor que foi implementado, foi possível verificar o funcionamento de uma requisição HTTP, como uma multithread é executado, e além disso, entender como os sites que tanto usamos hoje em dia são feitos, como ficam disponíveis, como o servidor recebe e manda a resposta para o Cliente, e assim desenvolvemos ainda mais as técnicas e aprendizados adquiridos durante a aula.

Obs: Link dos arquivos do trabalho armazenado no Github

https://github.com/Paulojct1/REDES_2019_1