Project 1: Web Server

Programmer: LaVonne Diller and Nate Kuhn

Course: CSC 431

Section: 1

Instructor: S. Lee

**Algorithm**

- The WebServer:

Creates a ServerSocket at port 8080

Establishes a socket

Listens for incoming request

Pass HttpRequest class to new thread when request is received

- HttpRequest:

Gets the request line of HTTP request message

Extracts file name from request line

Find & Open request file

Constructs response message with statusLine, contentType, and entityBody

Sends response message to client as bytes

Close streams and sockets

**Description of program**

The WebServer class creates a socket at port 8080 and waits indefinitely for an incoming HTTP request message. When the WebServer receives HTTP request message, it creates an instance of HttpRequest and passes the instance to a new thread. The HttpRequest class processes the HTTP request message by parsing the request line for the expected file type. Next, HttpRequest finds and opens the requested file. Then, it constructs a HTTP response message that encludes a status code, status message, and content type in its header. Finally, the HttpRequest class sends the Http response header and the body (the expected file) in bytes to the client via the socket. In addition, the HttpRequest class closes the streams and socket.

**Analysis of algorithm and output**

Entering: http://localhost:8080/test.html

Result: The expected html file, text.html, with jpg and gif images.

Entering: http://localhost:8080/file.jpg

Result: Shows nothing. Downloads the file instead.

Entering: http://localhost:8080/littleguy.gif

Result: Displays the gif image on the page.

Entering: http://localhost:8080/test2.html

Result: Not Found (The error page)

**Codes**

**import java.net.ServerSocket;**

**import java.net.Socket;**

**/\***

**\* Project 1: Web Server**

**\* Programmer: LaVonne Diller and Nate Kuhn**

**\* Course: CSC 431**

**\* Section: 1**

**\* Instructor: S. Lee**

**\*/**

**/\*\***

**\* A Web Server that listen to and handles incoming requests**

**\*/**

**public final class WebServer {**

**@SuppressWarnings("resource")**

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

**ServerSocket server = new ServerSocket(8080);**

**// Establish the listen socket.**

**Socket sock = new Socket();**

**// Process HTTP service requests in an infinite loop.**

**while(true) {**

**// Wait for an incoming request**

**sock = server.accept();**

**// Construct an object to process the HTTP request message.**

**HttpRequest request = new HttpRequest(sock);**

**// Create a new thread to process the request.**

**Thread thread = new Thread(request);**

**thread.start();**

**}**

**}**

**}**

**import** java.io.BufferedReader;

**import** java.io.DataOutputStream;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.InputStream;

**import** java.io.InputStreamReader;

**import** java.net.Socket;

**import** java.util.StringTokenizer;

/\*

\* Project 1: Web Server

\* Programmer: LaVonne Diller and Nate Kuhn

\* Course: CSC 431

\* Section: 1

\* Instructor: S. Lee

\*/

/\*\*

\* The class that handle the HTTP Request

\*/

**public** **final** **class** HttpRequest **implements** Runnable {

**final** **static** String ***CRLF*** = "\r\n";

Socket socket;

// Constructor

**public** HttpRequest (Socket socket) **throws** Exception {

**this**.socket = socket;

}

@Override

**public** **void** run() {

**try** {

processRequest();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

/\*\*

\* Processes HTTP Request message and builds HTTP Response message

\* **@throws** Exception

\*/

**private** **void** processRequest() **throws** Exception {

// Get a reference to the socket's input and output streams

InputStream is = socket.getInputStream();

DataOutputStream os = **new** DataOutputStream(socket.getOutputStream());

// Set up input Stream filters

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(is));

// Get the request line of the HTTP request message

String requestLine = br.readLine();

// Display the request line

System.***out***.println();

System.***out***.println(requestLine);

// Get and display the header lines.

String headerLine = **null**;

**while**((headerLine = br.readLine()).length() != 0) {

System.***out***.println(headerLine);

}

// Extract the filename from the request line.

StringTokenizer tokens = **new** StringTokenizer(requestLine);

tokens.nextToken(); // skip the method, which should be "GET"

String fileName = tokens.nextToken();

//Prepend a "." so that file request is within the current directory.

fileName = "." + fileName;

// Open the request file.

FileInputStream fis = **null**;

**boolean** fileExists = **true**;

**try** {

fis = **new** FileInputStream(fileName);

} **catch** (FileNotFoundException e) {

fileExists = **false**;

}

// Construct the response message.

String statusLine = **null**;

String contentTypeLine = **null**;

String entityBody = **null**;

**if** (fileExists) {

statusLine = "HTTP /1.0 200 OK" + ***CRLF***;

contentTypeLine = "Content-Type: " + contentType(fileName) + ***CRLF***;

} **else** {

statusLine = "HTTP /1.0 404 Not Found" + ***CRLF***;

contentTypeLine = "";

entityBody = "<!DOCTYPE html><HTML><HEAD><TITLE>Not Found</TITLE></HEAD><BODY>Not Found</BODY></HTML>";

}

// Sends the status line, content type line, and the entity body to the client.

os.writeBytes(statusLine);

os.writeBytes(contentTypeLine);

os.writeBytes(***CRLF***);

// Sends the Entity Body if exists

**if** (fileExists) {

*sendBytes*(fis, os);

fis.close();

} **else** {

os.writeBytes(entityBody);

}

// Close Streams and socket.

os.close();

br.close();

socket.close();

}

/\*\*

\* Sends bytes that is the HTTP Response message header & entity body

\* **@param** fis FileInputStream

\* **@param** os DataOutputStream

\* **@throws** Exception

\*/

**private** **static** **void** sendBytes(FileInputStream fis, DataOutputStream os) **throws** Exception {

// Construct a 1K buffer to hold bytes on their way to the socket.

**byte**[] buffer = **new** **byte**[1024];

**int** bytes = 0;

// Copy requested file into the socket's output stream.

**while**((bytes = fis.read(buffer)) != -1 ) {

os.write(buffer, 0, bytes);

}

}

/\*\*

\* Searches filename for the file type requested

\* and returns the contentType for the HTTP response message

\* **@param** fileName name of file requested

\* **@return** contentType

\*/

**private** String contentType(String fileName) {

**if**(fileName.endsWith(".htm") || fileName.endsWith(".html")) {

**return** "text/html";

}

**else** **if**(fileName.endsWith(".gif")) {

**return** "image/gif";

}

**else** **if**(fileName.endsWith(".jpeg")) {

**return** "image/jpeg";

}

**return** "application/octet-stream";

}

}