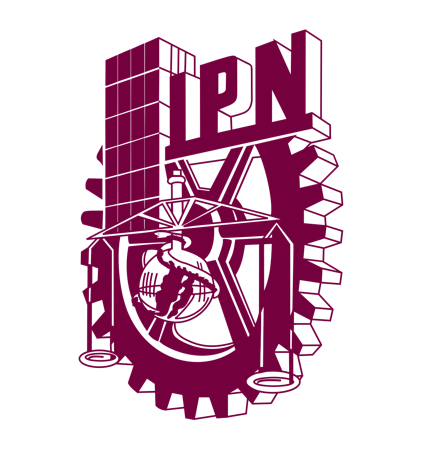
**INSTITUTO POLITECNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO.**

DÍAZ MEDINA JESÚS KAIMORTS

ESQUIVEL VALDEZ ALBERTO

VARGAR ROMERO ERICK EFRÁIN

**Aplicaciones para comunicaciones de red**

Axel Ernesto Moreno Cervantes

**Práctica 5.**

**Introducción**

Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente es renderizado por un navegador web. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se usa el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI. El término también se

emplea para referirse al ordenador.

El IETF (Internet Engineering Task Force) es un grupo de trabajo encargado de validar y aprobar todos los protocolos que se usan en Internet para garantizar la interoperabilidad entre aplicaciones en red independientemente de arquitecturas, sistemas operativos, lenguajes de programación, etc. Para lograr este objetivo desarrollan especificaciones llamadas RFC (Request For Comments). Una ventaja de contar con la especificación de un protocolo es el poder implementarla o modificar algunas de sus características basándonos en una implementación. En este caso nos gustaría implementar una

pequeña versión de un Servidor HTTP la cual se ajuste a la especificación de la RFC 2616.

**Desarrollo**

El servidor cuenta con el código para procesar solicitudes tipo GET, POST, HEAD y DELETE, además de ser capaz de procesar archivos y detectar errores en la petición.

Para realizar peticiones se utilizó un software llamado Postman, este nos ayuda a dirigir peticiones de cualquier tipo a la dirección especificada y nos muestra la respuesta del servidor.

Imagen que contiene captura de pantalla

Descripción generada con confianza muy alta

Fig. . Postman. Software utilizado para hacer peticiones

A continuación, se muestra el procesamiento de diferentes tipos de solicitudes del lado derl servidor:

Imagen que contiene captura de pantalla, monitor, ordenador

Descripción generada con confianza muy alta

Fig. 2. Procesamiento de una solicitud tipo GET

Imagen que contiene captura de pantalla, monitor, ordenador

Descripción generada con confianza muy alta

Fig. 3. Procesamiento de una solicitud tipo POST

Imagen que contiene captura de pantalla, ordenador, portátil, interior

Descripción generada con confianza muy alta

Fig. 4. Procesamiento de una solicitud tipo DELETE

Imagen que contiene captura de pantalla, ordenador, portátil, interior

Descripción generada con confianza muy alta

Fig. 5. Procesamiento de una solicitud tipo HEAD

A continuación se muestra el código utilizado para procesar las solicitudes:

1. **public** **class** ServidorWeb {
3. **public** **static** **final** **int** PUERTO = 8000;
4. ServerSocket ss;
6. **class** Manejador **extends** Thread {
8. **protected** Socket socket;
9. **protected** PrintWriter pw;
10. **protected** BufferedOutputStream bos;
11. **protected** BufferedReader br;
12. **protected** String FileName;
13. **protected** DataInputStream dis;
14. **protected** Cliente c;
16. **public** Manejador(Socket \_socket, Cliente c) **throws** Exception {
17. **this**.socket = \_socket;
18. **this**.c = c;
19. }
21. **public** **void** run() {
22. **try** {
23. br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));
24. bos = **new** BufferedOutputStream(socket.getOutputStream());
25. dis = **new** DataInputStream(socket.getInputStream());
26. pw = **new** PrintWriter(**new** OutputStreamWriter(bos));
27. String line = br.readLine();
28. StringBuilder head = **new** StringBuilder();
29. StringBuilder body = **new** StringBuilder();
30. **int** contentLength = 0;
32. System.out.println("Line: " + line);
34. **if** (line == **null**) {
35. pw.print("<html><head><title>Servidor WEB");
36. pw.print("</title><body bgcolor=\"#aaffff\"<br>Linea Vacia</br>");
37. pw.print("</body></html>");
38. socket.close();
39. **return**;
40. }
42. System.out.println("\nCliente Conectado desde: " + socket.getInetAddress());
43. System.out.println("Por el puerto: " + socket.getPort());
44. System.out.println("Datos: " + line + "\r\n\r\n");
46. **if** (line.toUpperCase().startsWith("POST")) {
47. System.out.println("POST");
48. //leemos el encabezado
49. **while** (!(line = br.readLine()).equals("")) {
50. head.append('\n' + line);
51. **final** String contentHeader = "Content-Length: ";
52. **if** (line.startsWith(contentHeader)) {
53. contentLength = Integer.parseInt(line.substring(contentHeader.length()));
54. }
56. }
57. System.out.println(head);
59. //Lectura de parametros
60. **int** c = 0;
61. **for** (**int** i = 0; i < contentLength; i++) {
62. c = br.read();
63. body.append((**char**) c);
64. }
66. System.out.println(body);
68. pw.println("HTTP/1.0 200 Okay");
69. pw.flush();
70. pw.println();
71. pw.flush();
72. pw.print("<html><head><title>SERVIDOR WEB");
73. pw.flush();
74. pw.print("</title></head><body bgcolor=\"#AACCFF\" text=\"FFFFFF\"><center><h1>Metodo POST<br>Parametros recibidos: </br></h1>");
75. pw.flush();
76. pw.print("<h3><b>" + body + "</b></h3>");
77. pw.flush();
78. pw.print("</center></body></html>");
79. pw.flush();
81. } **else** **if** (line.toUpperCase().startsWith("DELETE")) {
82. System.out.println("DELETE");
83. pw.println("HTTP/1.0 200 Okay");
84. **int** i = line.indexOf("/");
85. **int** f = line.indexOf(" ", i);
86. FileName = line.substring(i + 1, f);
88. File file = **new** File(FileName);
89. **if** (file.delete()) {
90. System.out.println("archivo eliminado");
91. } **else** {
92. System.out.println("El archivo no se pudo eliminar");
93. }
94. pw.flush();
95. } **else** **if** (line.toUpperCase().startsWith("HEAD")) {
96. System.out.println("HEAD");
97. line = br.readLine();
98. System.out.println(line);
99. pw.println("HTTP/1.0 200 Okay");
100. pw.flush();
101. } **else** **if** (line.indexOf("?") == -1) {
102. getArch(line);//obtiene el archivo a enviar
103. **if** (FileName.compareTo("") == 0) {
104. SendA("index.htm");
105. } **else** {
106. SendA(FileName);//envia el arhivo solicitado
107. }
108. System.out.println(FileName);
110. } **else** **if** (line.toUpperCase().startsWith("GET")) {
111. System.out.println("GET");
112. StringTokenizer tokens = **new** StringTokenizer(line, "?");
113. String req\_a = tokens.nextToken();
114. String req = tokens.nextToken();
115. System.out.println("Token1: " + req\_a + "\r\n\r\n");
116. System.out.println("Token2: " + req + "\r\n\r\n");
117. pw.println("HTTP/1.0 200 Okay");
118. pw.flush();
119. pw.println();
120. pw.flush();
121. pw.print("<html><head><title>SERVIDOR");
122. pw.flush();
123. pw.print("</title></head><body bgcolor=\"#AACCFF\" text=\"FFFFFF\"><center><h1>Metodo GET<br>Parametros Obtenidos..</br></h1>");
124. pw.flush();
125. pw.print("<h3><b>" + req + "</b></h3>");
126. pw.flush();
127. pw.print("</center></body></html>");
128. pw.flush();
129. } **else** {
130. pw.println("HTTP/1.0 501 Not Implemented");
131. pw.println();
132. }
133. pw.flush();
134. bos.flush();
135. } **catch** (Exception e) {
136. e.printStackTrace();
137. }
138. **try** {
139. socket.close();
140. } **catch** (Exception e) {
141. e.printStackTrace();
142. } **finally** {
143. c.deconectar();
144. }
145. }
147. **public** **void** getArch(String line) {
148. **int** i;
149. **int** f;
150. **if** (line.toUpperCase().startsWith("GET")) {
151. i = line.indexOf("/");
152. f = line.indexOf(" ", i);
153. FileName = line.substring(i + 1, f);
154. }
155. }
157. **public** **void** SendA(String fileName, Socket sc) {
158. //System.out.println(fileName);
159. **int** fSize = 0;
160. **byte**[] buffer = **new** **byte**[4096];
161. **try** {
162. DataOutputStream out = **new** DataOutputStream(sc.getOutputStream());
164. //sendHeader();
165. FileInputStream f = **new** FileInputStream(fileName);
166. **int** x = 0;
167. **while** ((x = f.read(buffer)) > 0) {
168. //      System.out.println(x);
169. out.write(buffer, 0, x);
170. }
171. out.flush();
172. f.close();
173. } **catch** (FileNotFoundException e) {
174. //msg.printErr("Transaction::sendResponse():1", "El archivo no existe: " + fileName);
175. } **catch** (IOException e) {
176. //          System.out.println(e.getMessage());
177. //msg.printErr("Transaction::sendResponse():2", "Error en la lectura del archivo: " + fileName);
178. }
180. }
182. **public** **void** SendA(String arg) {
183. String extension = arg.substring(arg.length() - 3, arg.length());
184. **try** {
185. **int** b\_leidos = 0;
186. BufferedInputStream bis2 = **new** BufferedInputStream(**new** FileInputStream(arg));
187. **byte**[] buf = **new** **byte**[1024];
188. **int** tam\_bloque = 0;
189. **if** (bis2.available() >= 1024) {
190. tam\_bloque = 1024;
191. } **else** {
192. bis2.available();
193. }
195. String tipo = extension.equals("jpg") ? "image/jpeg \n" : extension.equals("pdf") ? "application/pdf \n" : extension.equals("png") ? "image/png \n" : extension.equals("gif") ? "image/gif \n" : extension.equals("mp3") ? "audio/mpeg \n" : extension.equals("txt") ? "text/plain \n" : "text/html \n";
196. **int** tam\_archivo = bis2.available();
197. /\*\*
198. \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
199. \*/
200. String sb = "";
201. sb = sb + "HTTP/1.0 200 ok\n";
202. sb = sb + "Server: Servidor/1.0 \n";
203. sb = sb + "Date: " + **new** Date() + " \n";
204. sb = sb + "Content-Type: " + tipo;
205. sb = sb + "Content-Length: " + tam\_archivo + " \n";
206. sb = sb + "\n";
207. bos.write(sb.getBytes());
208. bos.flush();
210. /\*\*
211. \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
212. \*/
213. **while** ((b\_leidos = bis2.read(buf, 0, buf.length)) != -1) {
214. bos.write(buf, 0, b\_leidos);
216. }
217. bos.flush();
218. bis2.close();
220. } **catch** (Exception e) {
221. System.out.println(e.getMessage());
222. }
224. }
225. }
227. **public** ServidorWeb() **throws** Exception {
228. Cliente c = **new** Cliente();
229. System.out.println("Iniciando Servidor.......");
230. **this**.ss = **new** ServerSocket(PUERTO);
231. System.out.println("Servidor iniciado:---OK");
232. System.out.println("Esperando por Cliente....");
233. **for** (;;) {
234. c.getConexion();
235. Socket accept = ss.accept();
236. **new** Manejador(accept, c).start();
237. }
238. }
240. **public** **static** **void** main(String[] args) **throws** Exception {
241. ServidorWeb sWEB = **new** ServidorWeb();
242. }
244. }

**Conclusiones**

Esta práctica nos permitió entender el funcionamiento de los servidores HTTP y el como se procesan y funcionan los diferentes tipos de solicitudes existentes, así como el uso y utilidades de cada una.