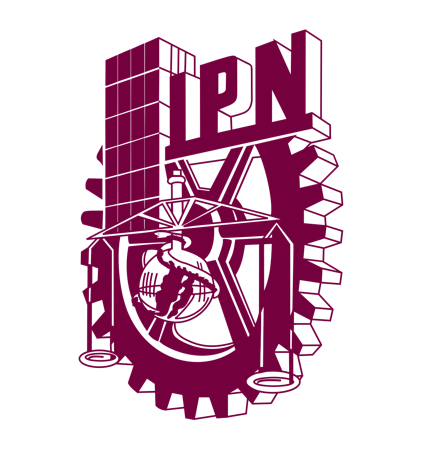
****

**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 6. Balanceador de Carga.**

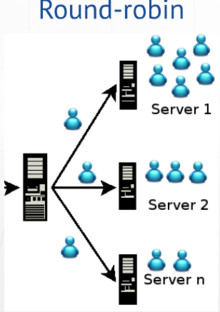
**Introducción.**

Un **balanceador de carga** fundamentalmente es un dispositivo de hardware o software que se pone al frente de un conjunto de servidores que atienden una aplicación, y este asigna las solicitudes que llegan de los clientes a los servidores usando algún algoritmo.

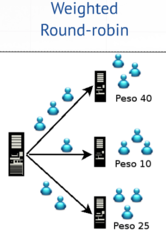
Los métodos de balanceo pueden ser: de petición, basado en sesión o de métodos.

Los algoritmos más comunes son:

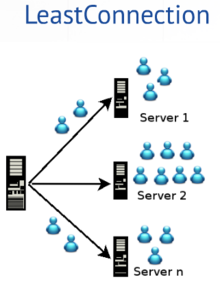
* [**Round-Robin**](https://es.wikipedia.org/wiki/Planificaci%C3%B3n_Round-robin)

Distribuye las peticiones de manera equitativa y racional. Empezaría por el primer servidor entregando las peticiones una a una hasta el último y vuelta a empezar.  
[](http://programacion.jias.es/wp-content/uploads/2016/09/Round-Robin.png)

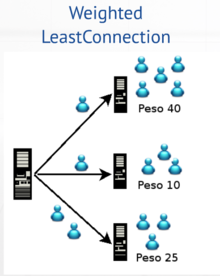
* [**Weighted Round-Robin**](https://en.wikipedia.org/wiki/Weighted_round_robin)

Se define un peso ponderado para cada uno de los servidores y se envían las peticiones en función de este peso. Es una mejora sobre el Round-Robin, ya que, por ejemplo, la máquina 1 puede manejar 4 veces la carga que soporta la máquina 2.  
[](http://programacion.jias.es/wp-content/uploads/2016/09/Weighted_Round-Robin.png)

* **Least Connection**

Las peticiones se hacen dependiendo del número de conexiones que tenga cada servidor en ese momento.  
[](http://programacion.jias.es/wp-content/uploads/2016/09/Least_Connection.png)

* **Weighted Least Connection**

Como en el caso anterior, las conexiones se entregan dependiendo del número de conexiones que tenga cada servidor en ese momento, pero dependiendo del peso ponderado asignado a cada una de las máquinas.  
[](http://programacion.jias.es/wp-content/uploads/2016/09/Weighted_LeastConnection.png)

* **Fastest**

Las peticiones se entregan al servidor que menor tiempo de respuesta tenga de cada uno de ellos. Esto es, al más rápido, como su nombre indica.

* **Observed**

Este algoritmo usa una lógica combinada de los algoritmos Least Connections y Fastest, haciendo un ranking de los servidores de acuerdo a una combinación de criterios como el número actual de conexiones y el tiempo de respuesta. Aquellos que tienen menor cantidad de conexiones y un tiempo de respuesta más rápido, reciben las nueva peticiones.

* **Predictive**

Usa la forma de ranking manejada por el algoritmo Observed, sin embargo, en este caso se analiza la tendencia del ranking, determinando qué servidores pueden mejorar o empeorar su rendimiento. Aquellos que tengan una tendencia de mejora, recibirán mayor cantidad de conexiones.

**Desarrollo.**

Para esta práctica se realizará una aplicación que nos permita hacer la descarga de archivos encontrados en diferentes computadoras. Cada una de ellas funcionara como cliente y servidor.

Se realizarán tres acciones en tres hilos, el primero consiste en que, mediante sockets multicast, se conectarán varios servidores a un mismo grupo, esto servirá para que, al buscar un archivo a descargar, este archivo se busque en todos servidores que se encuentren conectados al grupo, que es en donde entra el segundo hilo que con RMI se mandará la búsqueda correspondiente a cada uno de los servidores conectados que nos va a regresar una lista con todas las coincidencias encontradas, y finalmente el tercer hilo que se encargará de descargar los o el archivo seleccionado.

**Código.**

1. **package** sources;
2. **import** java.nio.channels.\*;
3. **import** java.nio.\*;
4. **import** java.net.\*;
5. **import** java.util.Iterator;
6. **import** java.io.\*;
7. **import** java.util.\*;
9. **public** **class** AdministradorHttp {
10. **public** **static** **void** main(String[] args){
11. **int** pto=9999;
12. **try**{
13. ArrayList<Integer> servidores=**new** ArrayList<Integer>();
14. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
15. System.out.println("Numero de Servidores:");
16. **int** nServidores=Integer.parseInt(br.readLine());
17. **for** (**int** i=0;i<nServidores ; i++) {
18. System.out.println("Dame el puerto para el servidor"+(i+1));
19. **int** puerto=Integer.parseInt(br.readLine());
20. servidores.add(puerto);
21. }
22. **int** puerto1=-1,posicion=0;
23. String respuesta="";
24. ServerSocketChannel s = ServerSocketChannel.open();
25. s.configureBlocking(**false**);
26. s.setOption(StandardSocketOptions.SO\_REUSEADDR, **true**);
27. s.socket().bind(**new** InetSocketAddress(pto));
28. Selector sel = Selector.open();
29. s.register(sel,SelectionKey.OP\_ACCEPT);
30. GestorServidores serv=**new** GestorServidores(3);
31. System.out.println("Servicio iniciado..esperando clientes..");
32. **while**(**true**){
33. sel.select();
34. Iterator<SelectionKey>it= sel.selectedKeys().iterator();
35. **while**(it.hasNext()){
36. SelectionKey k = (SelectionKey)it.next();
37. it.remove();
38. **if**(k.isAcceptable()){
39. SocketChannel cl = s.accept();
40. System.out.println("Cliente conectado desde->"+cl.socket().getInetAddress().getHostAddress()+":"+cl.socket().getPort());
41. cl.configureBlocking(**false**);
42. cl.register(sel,SelectionKey.OP\_READ);
43. puerto1=servidores.get(posicion);
44. Iterator<Integer> nombreIterator = servidores.iterator();
45. **while**(nombreIterator.hasNext()){
46. **int** elemento = nombreIterator.next();
47. System.out.print(elemento+" / ");
48. }
49. **continue**;
50. }//if
51. **if**(k.isReadable()){
52. SocketChannel ch =(SocketChannel)k.channel();
53. ByteBuffer b = ByteBuffer.allocate(2000);
54. b.clear();
55. **int** n = ch.read(b);
57. **if** ( n > -1) {
58. String msj = **new** String(b.array(),0,n);
59. respuesta="";
60. System.out.println("Peticion Recibida: "+msj);
61. ConexionServidores cs=**new** ConexionServidores(serv,puerto1,puerto1,msj);
62. cs.start();
63. cs.join();
64. respuesta=cs.getMensaje();
65. respuesta=respuesta.replace("%n","\n");
66. ByteBuffer b2=ByteBuffer.wrap(respuesta.getBytes());
67. ch.write(b2);
68. k.interestOps(SelectionKey.OP\_WRITE);
69. **continue**;
70. }
71. }//if\_readable
72. **if**(k.isWritable()){
73. **if** (!respuesta.equals("")) {
74. System.out.println("\n\nRespuesta a enviar:\n "+respuesta);
75. SocketChannel ch2 = (SocketChannel)k.channel();
76. ByteBuffer b = ByteBuffer.wrap(respuesta.getBytes());
77. ch2.write(b);
78. k.interestOps(SelectionKey.OP\_READ);
79. System.out.println("La respuesta fue enviada");
80. servidores.add(puerto1);
81. posicion++;
82. puerto1=-1;
83. respuesta="";
84. **continue**;
85. }
86. }//if\_writable
87. }//while
88. }//while
89. }**catch**(Exception e){
90. e.printStackTrace();
91. }//catch
92. }//main
93. }

Adminsitrador de las conexiones a servidores

1. **package** sources;
2. **import** java.util.Random;
3. **import** java.net.\*;
4. **import** java.io.\*;
5. **class** ConexionServidores **extends** Thread {
6. **private** GestorServidores r;
7. **private** **int** socketID,puerto;
8. **private** String mensaje,respuesta,disponibilidad;
9. **public** ConexionServidores(GestorServidores r, **int** socketID,**int** puerto,String mensaje) {
10. **this**.r = r;
11. **this**.socketID = socketID;
12. **this**.mensaje=mensaje;
13. **this**.puerto=puerto;
14. respuesta="";
15. disponibilidad="True";
17. }
18. **public** **void** run() {
19. r.getConexion(**this**.socketID); // Get a Conection
20. disponibilidad="False";
21. **try** {
22. InetAddress srv = InetAddress.getByName("127.0.0.1");
23. Socket cl = **new** Socket(srv,puerto);
24. PrintWriter pw = **new** PrintWriter(**new** OutputStreamWriter(cl.getOutputStream()));
25. BufferedReader br = **new** BufferedReader(**new** InputStreamReader(cl.getInputStream()));
26. System.out.println("El socket #"+socketID+" esperara 2 s.");
27. sleep(2000);
28. pw.println(mensaje);
29. pw.flush();
30. **for**(String linea = br.readLine(); linea != **null**; linea = br.readLine()){
31. respuesta+=linea;
32. }
33. pw.close();
34. br.close();
35. cl.close();
36. }
37. **catch**(Exception e) {
38. e.printStackTrace();
39. }
40. **finally** {
41. r.liberaConexion(**this**.socketID);
42. }
43. }
45. **public** String getMensaje(){
46. **return** respuesta;
47. }
49. **public** **int** getPuerto(){
50. **return** puerto;
51. }
53. **public** String getDisponibilidad(){
54. **return** disponibilidad;
55. }
57. **public** **void** setDisponibilidad(String disponibilidad){
58. **this**.disponibilidad=disponibilidad;
59. }
60. }

Gestor de conexiones

1. **package** sources;
2. **import** java.util.concurrent.Semaphore;
4. **public** **class** GestorServidores {
5. **private** Semaphore servidores;
6. **public** GestorServidores(**int** servidoresCount) {
7. // Create a semaphore using number of servidores we have
8. **this**.servidores = **new** Semaphore(servidoresCount);
9. }
10. **public** **void** getConexion(**int** socketID) {
11. System.out.println("getConexion: "+socketID);
12. **try** {
13. System.out.println("El socket #" + socketID + " esta intentando obtener una conexion con un servidor.");
14. // Acquire a permit for a conection
15. servidores.acquire();
16. System.out.println("El socket #" + socketID + " consiguio conectarse con un servidor.");
17. }
18. **catch** (InterruptedException e) {
19. e.printStackTrace();
20. }
21. }
22. **public** **void** liberaConexion(**int** socketID) {
23. System.out.println("El socket #" + socketID + " libero la conexion.");
24. servidores.release();
25. }
26. /\*public static void main(String[] args) {
27. // Create a restaurant with two dining servidores
28. Restaurante r = new Restaurante(2);
29. // Create five customers
30. for (int i = 1; i <= 5; i++) {
31. Cliente c = new Cliente(r, i);
32. c.start();
33. }
34. }\*/
35. }

HttpRequest

1. **package** sources;
2. **import** java.util.\*;
3. **import** java.util.regex.Pattern;
5. **public** **class** HttpRequest {
6. **private** String peticion;
7. **private** Map<String,String> encabezado;
9. **public** HttpRequest(String peticion){
10. **this**.peticion=peticion;
11. encabezado=**new** HashMap<String,String>();
12. getEncabezado(peticion);
13. }
15. **public** String getTipo(){
16. **switch** (peticion.substring(0,peticion.indexOf(" "))) {
17. **case** "GET":
18. **return** "GET";
19. **case** "POST":
20. **return** "POST";
21. **case** "HEAD":
22. **return** "HEAD";
23. **case** "DELETE":
24. **return** "DELETE";
25. **default**:
26. **return** "HTTP/1.0 501 Not Implemented";
27. }
28. }
30. **public** **void** getEncabezado(String linea){
31. String [] headerParams = linea.split(Pattern.quote("\n"));
32. **for**(**int** i = 0; i<headerParams.length;i++){
33. **try**{
34. **if**(i==0){
35. String headerValues[] = headerParams[i].split(Pattern.quote(" "));
36. **if**(headerValues[0].length() > 0){
37. System.out.println("header REC: "+headerValues[0]+" value: "+headerValues[1]);
38. encabezado.put(headerValues[0], headerValues[1]);
39. }
40. }**else**{
41. **if**(headerParams[i].length() > 0){
42. String headerValues[] = headerParams[i].split(Pattern.quote(":"));
43. encabezado.put(headerValues[0], headerValues[1]);
44. }
45. }
46. }**catch**(ArrayIndexOutOfBoundsException e){
48. }
49. }
50. }
52. **public** **void** imprimeEncabezado(){
53. Iterator iterador=encabezado.keySet().iterator();
54. **while**(iterador.hasNext()){
55. String parametro=(String)iterador.next();
56. System.out.println(parametro+": "+encabezado.get(parametro));
57. }
58. }
60. **public** String getValue(String header){
61. //System.out.println("Header: "+header);
62. **if**(encabezado.containsKey(header)){
63. //System.out.println("Value: "+encabezado.get(header));
64. **return** encabezado.get(header);
65. }**else**{
66. **return** "-1";
67. }
68. }
70. **public** String getUri(){
71. String parametrosEncabezado[]=peticion.split(Pattern.quote("\n"));
72. String valorEncabezados[]=parametrosEncabezado[0].split(Pattern.quote(" "));
73. **return** valorEncabezados[1].substring(1,valorEncabezados[1].length());
74. }
75. }

Manejador

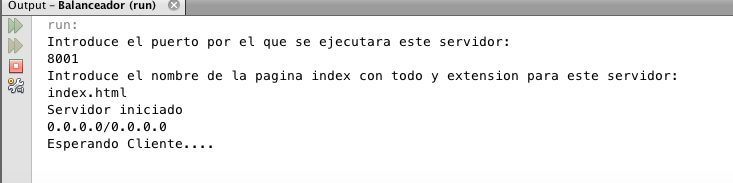
1. **package** sources;
2. **import** java.net.\*;
3. **import** java.io.\*;
4. **import** java.util.\*;
5. **import** java.util.logging.Level;
6. **import** java.util.logging.Logger;
7. **import** java.util.regex.Pattern;
9. **public** **class** Manejador **extends** Thread **implements** Metodos{
11. **protected** Socket socket;
12. **protected** BufferedInputStream request;
13. **protected** BufferedOutputStream bos;
14. **protected** PrintWriter response;
15. **protected** String FileName;
16. **protected** Map<String, String> extensiones;
17. **protected** Map<String, String> archivos;
18. **protected** **int** estatusCode;
19. **protected** **byte** permiso;
20. **protected** **boolean** payload;
21. **protected** **long** max\_paylod;
22. **protected** **byte** estado\_servidor;
23. **protected** String index;
25. **public** Manejador(Socket \_socket,String index) **throws** Exception {
26. //TODO: 429 Too many requests
27. //TODO: 503 Service unavailable
28. **this**.index=index;
29. socket = \_socket;
30. request = **new** BufferedInputStream(**new** DataInputStream(socket.getInputStream()));
31. bos = **new** BufferedOutputStream(socket.getOutputStream());
32. response = **new** PrintWriter(**new** OutputStreamWriter(bos));
33. estatusCode = 200;
34. payload = **true**;
35. max\_paylod = 1500; //1500 para no mandar imagen
36. estado\_servidor = 1; //2 - Mantenimiento
38. extensiones = **new** HashMap<String, String>();
39. extensiones.put("aac", "audio/aac");
40. extensiones.put("abw", "application/x-abiword");
41. extensiones.put("arc", "application/octet-stream");
42. extensiones.put("avi", "video/x-msvideo");
43. extensiones.put("azw", "application/vnd.amazon.ebook");
44. extensiones.put("bin", "application/octet-stream");
45. extensiones.put("bz", "application/x-bzip");
46. extensiones.put("bz2", "application/x-bzip2");
47. extensiones.put("csh", "application/x-csh");
48. extensiones.put("css", "text/css");
49. extensiones.put("csv", "text/csv");
50. extensiones.put("epub", "application/epub+zip");
51. extensiones.put("ico", "image/x-icon");
52. extensiones.put("ics", "text/calendar");
53. extensiones.put("jar", "application/java-archive");
54. extensiones.put("js", "application/javascript");
55. extensiones.put("json", "application/json");
56. extensiones.put("mid", "audio/midi");
57. extensiones.put("midi", "audio/midi");
58. extensiones.put("mpkg", "application/vnd.apple.installer+xml");
59. extensiones.put("odp", "application/vnd.oasis.opendocument.presentation");
60. extensiones.put("ods", "application/vnd.oasis.opendocument.spreadsheet");
61. extensiones.put("odt", "application/vnd.oasis.opendocument.text");
62. extensiones.put("oga", "audio/ogg");
63. extensiones.put("ogv", "video/ogg");
64. extensiones.put("ogx", "application/ogg");
65. extensiones.put("ppt", "application/vnd.ms-powerpoint");
66. extensiones.put("rar", "application/x-rar-compressed");
67. extensiones.put("rtf", "application/rtf");
68. extensiones.put("sh", "application/x-sh");
69. extensiones.put("svg", "image/svg+xml");
70. extensiones.put("swf", "application/x-shockwave-flash");
71. extensiones.put("tar", "application/x-tar");
72. extensiones.put("tif", "image/tiff");
73. extensiones.put("tiff", "image/tiff");
74. extensiones.put("ttf", "font/ttf");
75. extensiones.put("vsd", "application/vnd.visio");
76. extensiones.put("wav", "audio/x-wav");
77. extensiones.put("weba", "audio/webm");
78. extensiones.put("webm", "video/webm");
79. extensiones.put("webp", "image/webp");
80. extensiones.put("woff", "font/woff");
81. extensiones.put("woff2", "font/woff2");
82. extensiones.put("xhtml", "application/xhtml+xml");
83. extensiones.put("xls", "application/vnd.ms-excel");
84. extensiones.put("xml", "application/xml");
85. extensiones.put("xul", "application/vnd.mozilla.xul+xml");
86. extensiones.put("zip", "application/zip");
87. extensiones.put("3gp", "video/3gpp");
88. extensiones.put("3g2", "video/3gpp2");
89. extensiones.put("7z", "application/x-7z-compressed");
91. extensiones.put("jpg", "image/jpeg");
92. extensiones.put("jpge", "image/jpeg");
93. extensiones.put("doc", "application/msword");
94. extensiones.put("gif", "image/gif");
95. extensiones.put("mpeg", "video/mpeg");
96. extensiones.put("html", "text/html");
97. extensiones.put("htm", "text/html");
98. extensiones.put("pdf", "application/pdf");
100. archivos = **new** HashMap<String, String>();
101. /\*Implementar rutina para permisos\*/
102. File publico = **new** File(".");
103. **for**(File archivo: publico.listFiles()){
104. //En este servidor los archivos xml requieren autorización, cambiar configuración
105. **if**(archivo.getName().contains("xml")){
106. archivos.put(archivo.getName(), "folder");
107. }**else** **if**(archivo.getName().contains("mf")){
108. //En el servidor solo se pueden subir archivos .mf, pero no se pueden ver
109. archivos.put(archivo.getName(),"post");
110. }**else**{
111. archivos.put(archivo.getName(), "public");
112. }
113. }
114. }
116. @Override
117. **public** **void** run() {
118. **try** {
119. //Configure maximum request size in this array
120. **byte** b[] = **new** **byte**[1024];
121. request.read(b);
122. String lineRequest = **new** String(b);
123. System.out.println("\nCliente Conectado desde: " + socket.getInetAddress());
124. System.out.println("Por el puerto: " + socket.getPort());
125. System.out.println("Request: " + lineRequest );
127. HttpRequest httpRequest = **new** HttpRequest(lineRequest);
128. //GET
129. **if**(lineRequest.toUpperCase().startsWith("GET")){
130. doGet(httpRequest);
131. //HEAD
132. }**else** **if**(lineRequest.toUpperCase().startsWith("HEAD")){
133. doHead(httpRequest);
134. //POST
135. }**else** **if**(lineRequest.toUpperCase().startsWith("POST")){
136. doPost(httpRequest);
137. //NOT METHOD FOUND 501
138. }**else** **if**(lineRequest.toUpperCase().startsWith("DELETE")){
139. doDelete(httpRequest);
140. //NOT METHOD FOUND 501
141. }**else**{
142. response.println("HTTP/1.0 501 Not Implemented");
143. response.println();
144. }
145. response.flush();
146. bos.flush();
147. //keep-alive
148. socket.close();
149. } **catch** (IOException e) {
150. System.err.println(e.toString());
151. }
152. }//run
154. @Override
155. **public** **void** doGet(HttpRequest httpRequest) {
156. //Validaciones de URI
157. String header;
158. String msj = " OK";
159. String URI = httpRequest.getUri();
160. **if**(URI.equals("")){
161. URI= index;
162. }
163. String headerResponse = "";
164. BufferedInputStream bis2;
165. **int** contentSize = 0;
166. **try** {
167. File f = **new** File(URI);
168. //HTTP 400 - Accepted for processing, but not completed
169. **if**(!f.exists()) {
170. estatusCode = 404;
171. URI = "404.html";
172. }**else**{
173. **if**(!archivos.get(f.getName()).equals("public")){
174. permiso = 2;
175. }
176. **if**(archivos.get(f.getName()).equals("post")){
177. permiso = 3;
178. }
179. }
181. bis2 = **new** BufferedInputStream(**new** FileInputStream(URI));
182. bis2.available();
183. contentSize = bis2.available();
184. **if**(contentSize>max\_paylod){
185. payload = **false**;
186. estatusCode = 413;
187. msj = "El payload excede el size de: "+max\_paylod;
188. }
189. //HTTP 200 Accepted and processing
190. headerResponse = headerResponse + "HTTP/1.0 -estatusCode- %n";
191. headerResponse = headerResponse + "Server: Servidor Maincra %n";
192. headerResponse = headerResponse + "Date: " + **new** Date() + " %n";
193. headerResponse = headerResponse + "Content-Length: " + contentSize + " %n";
194. String[] partesFileName = URI.split(Pattern.quote("."));
196. header = httpRequest.getValue("Accept");
197. //Accept -> Content-type
198. **if** (extensiones.containsKey(partesFileName[(partesFileName.length - 1)])) {
199. **if**(!header.equals("-1") && (header.contains(extensiones.get(partesFileName[(partesFileName.length - 1)])) || header.contains("\*/\*"))){
200. headerResponse = headerResponse + "Content-Type: " +
201. extensiones.get(partesFileName[(partesFileName.length - 1)]) + " %n";
202. }**else**{
203. //content type incorrecto
204. System.out.println("no conozco ese content-type");
205. }
206. }**else**{
207. estatusCode = 400;
208. msj="Encabezado incorrecto";
209. }
210. //Accept-Language -> Content-Language
211. header = httpRequest.getValue("Accept-Language");
212. **if**(!header.equals("-1")){
213. //ingles
214. **if**(header.contains("en-US") || header.contains("en")){
215. headerResponse = headerResponse + "Content-Language: en %n";
216. }**else**{
217. headerResponse = headerResponse + "Content-Language: es %n";
218. }
219. }**else**{
220. estatusCode = 400;
221. msj="Encabezado incorrecto";
222. }
223. //Connection -> Connection
224. header = httpRequest.getValue("Connection");
225. **if**(!header.equals("-1")){
226. **if**(header.contains("keep-alive")){
227. headerResponse = headerResponse + "Connection: keep-alive %n";
228. }**else** **if**(header.contains("close")){
229. headerResponse = headerResponse + "Connection: close %n";
230. }
231. }**else**{
232. headerResponse = headerResponse + "Connection: close %n";
233. estatusCode = 400;
234. msj="Encabezado incorrecto";
235. }
236. //Authorization
237. **if**(permiso == 2){ //se necesita Autorización
238. header = httpRequest.getValue("Authorization");
239. **if**(!header.equals("-1")){ //check credentials
240. **if**(header.contains("Basic")){
241. payload = **true**;
242. }**else**{
243. payload = **false**;
244. estatusCode = 403;
245. msj = "Forbidden";
246. }
247. }**else**{
248. headerResponse = headerResponse + "WWW-Authenticate: Basic realm=\"Manda tus credenciales\"  %n";
249. msj = "Autenticate";
250. estatusCode = 401; //send credentials
251. }
252. }**else** **if**(permiso == 3){
253. estatusCode = 405;
254. payload=**false**;
255. msj="Método no permitido solo POST";
256. }
257. //esta linea completa un http response
258. headerResponse = headerResponse + "%n";
259. headerResponse = headerResponse.replace("-estatusCode-", Integer.toString(estatusCode)+msj);
260. bos.write(headerResponse.getBytes());
261. bos.flush();
262. **if**(payload==**true**){
263. **byte**[] buf = **new** **byte**[1024];
264. **int** b\_leidos = 0;
265. **while** ((b\_leidos = bis2.read(buf, 0, buf.length)) != -1) {
266. bos.write(buf, 0, b\_leidos);
267. }
268. bos.flush();
269. }
270. System.out.println("%n"+headerResponse);
272. } **catch** (IOException e) {
273. //Implement 505
274. }
275. }
277. @Override
278. **public** **void** doPost(HttpRequest httpRequest) {
279. //Validaciones de URI
280. String header;
281. String msj = " OK";
282. String URI = httpRequest.getUri();
283. **if**(URI.equals("")){
284. URI= index;
285. }
286. String headerResponse = "";
287. BufferedInputStream bis2;
288. **int** contentSize = 0;
289. **try** {
290. File f = **new** File(URI);
291. //HTTP 400 - Accepted for processing, but not completed
292. **if**(!f.exists()) {
293. estatusCode = 404;
294. URI = "404.html";
295. }**else**{
296. **if**(!archivos.get(f.getName()).equals("public")){
297. permiso = 2;
298. }
299. **if**(archivos.get(f.getName()).equals("post")){
300. permiso = 3;
301. }
302. }
304. bis2 = **new** BufferedInputStream(**new** FileInputStream(URI));
305. bis2.available();
306. contentSize = bis2.available();
307. **if**(contentSize>max\_paylod){
308. payload = **false**;
309. estatusCode = 413;
310. msj = "El payload excede el size de: "+max\_paylod;
311. }
312. //HTTP 200 Accepted and processing
313. headerResponse = headerResponse + "HTTP/1.0 -estatusCode- %n";
314. headerResponse = headerResponse + "Server: Servidor Maincra %n";
315. headerResponse = headerResponse + "Date: " + **new** Date() + " %n";
316. headerResponse = headerResponse + "Content-Length: " + contentSize + " %n";
317. String[] partesFileName = URI.split(Pattern.quote("."));
319. header = httpRequest.getValue("Accept");
320. //Accept -> Content-type
321. **if** (extensiones.containsKey(partesFileName[(partesFileName.length - 1)])) {
322. **if**(!header.equals("-1") && (header.contains(extensiones.get(partesFileName[(partesFileName.length - 1)])) || header.contains("\*/\*"))){
323. headerResponse = headerResponse + "Content-Type: " +
324. extensiones.get(partesFileName[(partesFileName.length - 1)]) + " %n";
325. }**else**{
326. //content type incorrecto
327. System.out.println("no conozco ese content-type");
328. }
329. }**else**{
330. estatusCode = 400;
331. msj="Encabezado incorrecto";
332. }
333. //Accept-Language -> Content-Language
334. header = httpRequest.getValue("Accept-Language");
335. **if**(!header.equals("-1")){
336. //ingles
337. **if**(header.contains("en-US") || header.contains("en")){
338. headerResponse = headerResponse + "Content-Language: en %n";
339. }**else**{
340. headerResponse = headerResponse + "Content-Language: es %n";
341. }
342. }**else**{
343. estatusCode = 400;
344. msj="Encabezado incorrecto";
345. }
346. //Connection -> Connection
347. header = httpRequest.getValue("Connection");
348. **if**(!header.equals("-1")){
349. **if**(header.contains("keep-alive")){
350. headerResponse = headerResponse + "Connection: keep-alive %n";
351. }**else** **if**(header.contains("close")){
352. headerResponse = headerResponse + "Connection: close %n";
353. }
354. }**else**{
355. headerResponse = headerResponse + "Connection: close %n";
356. estatusCode = 400;
357. msj="Encabezado incorrecto";
358. }
359. //Authorization
360. **if**(permiso == 2){ //se necesita Autorización
361. header = httpRequest.getValue("Authorization");
362. **if**(!header.equals("-1")){ //check credentials
363. **if**(header.contains("Basic")){
364. payload = **true**;
365. }**else**{
366. payload = **false**;
367. estatusCode = 403;
368. msj = "Forbidden";
369. }
370. }**else**{
371. headerResponse = headerResponse + "WWW-Authenticate: Basic realm=\"Manda tus credenciales\"  %n";
372. msj = "Autenticate";
373. estatusCode = 401; //send credentials
374. }
375. }**else** **if**(permiso == 3){
376. estatusCode = 405;
377. payload=**false**;
378. msj="Método no permitido solo POST";
379. }
380. //esta linea completa un http response
381. headerResponse = headerResponse + "%n";
382. headerResponse = headerResponse.replace("-estatusCode-", Integer.toString(estatusCode)+msj);
383. bos.write(headerResponse.getBytes());
384. bos.flush();
385. **if**(payload==**true**){
386. **byte**[] buf = **new** **byte**[1024];
387. **int** b\_leidos = 0;
388. **while** ((b\_leidos = bis2.read(buf, 0, buf.length)) != -1) {
389. bos.write(buf, 0, b\_leidos);
390. }
391. bos.flush();
392. }
394. } **catch** (IOException e) {
395. //Implement 505
396. }
397. }
399. @Override
400. **public** **void** doPut(HttpRequest httpRequest) {
401. **throw** **new** UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
402. }
404. @Override
405. **public** **void** doHead(HttpRequest httpRequest) {
406. **try**{
407. String URI = httpRequest.getUri();
408. **if**(URI.equals("")){
409. URI= index;
410. }
411. String headerResponse = "";
412. BufferedInputStream bis2;
413. File f = **new** File(URI);
414. **int** contentSize = 0;
415. //HTTP 202 - Accepted for processing, but not completed
416. **if**(!f.exists() || f.isDirectory()) {
417. headerResponse = headerResponse + "HTTP/1.0 202 NOT FOUND%n";
418. headerResponse = headerResponse + "Server: Servidor Maincra %n";
419. headerResponse = headerResponse + "Date: " + **new** Date() + " %n";
420. headerResponse = headerResponse + "Content-Type: text/html %n";
421. headerResponse = headerResponse + "Content-Length: " + contentSize + " %n";
422. headerResponse = headerResponse + "%n";
423. bos.write(headerResponse.getBytes());
424. bos.flush();
425. **return**;
426. }**else**{
427. bis2 = **new** BufferedInputStream(**new** FileInputStream(URI));
428. bis2.available();
429. contentSize = bis2.available();
430. headerResponse = headerResponse + "HTTP/1.0 200 OK%n";
431. }
432. //HTTP 200 Accepted and processing
433. headerResponse = headerResponse + "Server: Servidor Maincra %n";
434. headerResponse = headerResponse + "Date: " + **new** Date() + " %n";
435. headerResponse = headerResponse + "Content-Length: " + contentSize + " %n";
436. String[] partesFileName = URI.split(Pattern.quote("."));
438. //Content-type
439. **if** (extensiones.containsKey(partesFileName[(partesFileName.length - 1)])) {
440. headerResponse = headerResponse + "Content-Type: " +
441. extensiones.get(partesFileName[(partesFileName.length - 1)]) + " %n";
442. }
443. //Accept-Language
444. String language = httpRequest.getValue("Accept-Language");
445. **if**(!language.equals("-1")){
446. **if**(language.contains("en-US") || language.contains("en")){
447. headerResponse = headerResponse + "Content-Language: en %n";
448. }
449. }
450. //esta linea completa un http response
451. headerResponse = headerResponse + "%n";
452. bos.write(headerResponse.getBytes());
453. bos.flush();
454. }**catch**(Exception e){
456. }
457. }
459. @Override
460. **public** **void** doDelete(HttpRequest httprequest) {
461. //httprequest.imprimeEncabezado();
462. String URI = httprequest.getUri();
463. String headerrespuesta = "";
464. **if**(URI.equals("")){
465. URI=index;
466. }
467. File f = **new** File(URI);
468. **if**(f.exists() && f.isFile()){
469. **boolean** accion=f.delete();
470. **if**(accion){
471. headerrespuesta="HTTP/1.1 200 OK %n";
472. headerrespuesta = headerrespuesta + "Server: Servidor Maincra %n";
473. headerrespuesta = headerrespuesta + "Date: " + **new** Date() + " %n";
474. String[] partesFileName = URI.split(Pattern.quote("."));
476. //Content-type
477. **if** (extensiones.containsKey(partesFileName[(partesFileName.length - 1)])) {
478. headerrespuesta = headerrespuesta + "Content-Type: " +
479. extensiones.get(partesFileName[(partesFileName.length - 1)]) + " %n";
480. }
481. headerrespuesta = headerrespuesta + "Content-Length: "+f.length()+"%n";
482. headerrespuesta = headerrespuesta + "%n";
483. }**else**{
484. headerrespuesta="HTTP/1.1 204 NO CONTENT %n";
485. headerrespuesta = headerrespuesta + "Server: Servidor Maincra %n";
486. headerrespuesta = headerrespuesta + "Date: " + **new** Date() + " %n";
487. String[] partesFileName = URI.split(Pattern.quote("."));
488. **if** (extensiones.containsKey(partesFileName[(partesFileName.length - 1)])) {
489. headerrespuesta = headerrespuesta + "Content-Type: " +
490. extensiones.get(partesFileName[(partesFileName.length - 1)]) + " %n";
491. }
492. headerrespuesta = headerrespuesta + "Content-Length: "+f.length()+"%n";
493. headerrespuesta = headerrespuesta + "%n";
494. }
495. }**else**{
496. headerrespuesta="HTTP/1.1 404 NOT FOUND %n";
497. headerrespuesta = headerrespuesta + "Server: Servidor Maincra %n";
498. headerrespuesta = headerrespuesta + "Date: " + **new** Date() + " %n";
499. headerrespuesta = headerrespuesta + "Content-Type: NO FOUND%n";
500. headerrespuesta = headerrespuesta + "Content-Length: 0%n";
501. headerrespuesta = headerrespuesta + "%n";
502. }
504. **try**{
505. bos.write(headerrespuesta.getBytes());
506. bos.flush();
507. }**catch**(Exception e){}
508. }
510. @Override
511. **public** **void** doConnect(HttpRequest httpRequest) {
512. **throw** **new** UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
513. }
515. @Override
516. **public** **void** doOptions(HttpRequest httpRequest) {
517. **throw** **new** UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
518. }
520. @Override
521. **public** **void** doTrace(HttpRequest httpRequest) {
522. **throw** **new** UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
523. }

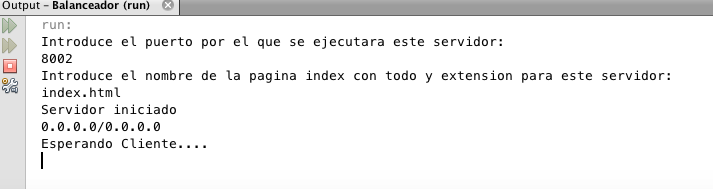
526. }

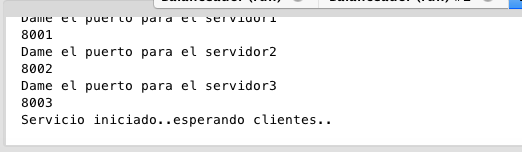
Servidor principal

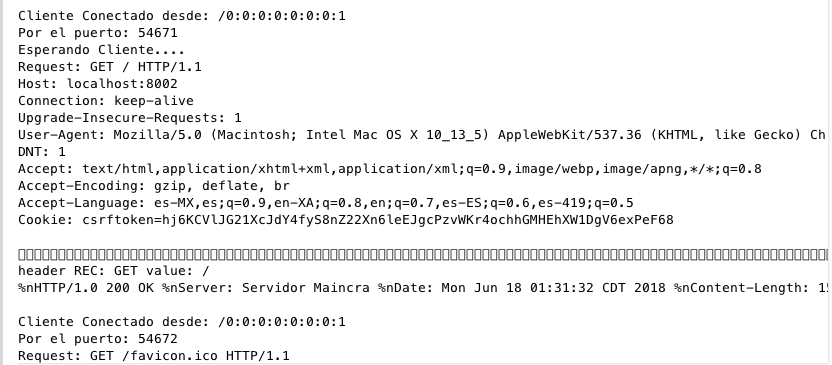
1. **package** sources;
2. **import** java.net.\*;
3. **import** java.io.\*;
4. **import** java.util.\*;
5. **import** java.util.logging.Level;
6. **import** java.util.logging.Logger;
7. **import** java.util.regex.Pattern;
9. **public** **class** ServidorPrincipal {
11. **public** **int** pto;
12. **public** ServerSocket ss;
14. **public** ServidorPrincipal(**int** puerto,String index) **throws** Exception {
15. pto=puerto;
16. **this**.ss = **new** ServerSocket(pto);
17. System.out.println("Servidor iniciado");
18. System.out.println(ss.getInetAddress());
19. **for** (;;) {
20. System.out.println("Esperando Cliente....");
21. Socket accept = ss.accept();
22. **new** Manejador(accept,index).start();
23. }
24. }
26. **public** **static** **void** main(String[] args) **throws** Exception {
27. **int** puerto=0;
28. String index="";
29. **try**{
30. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
31. System.out.println("Introduce el puerto por el que se ejecutara este servidor:");
32. puerto=Integer.parseInt(br.readLine());
33. System.out.println("Introduce el nombre de la pagina index con todo y extension para este servidor:");
34. index=br.readLine();
35. }**catch** (Exception e) {
37. }
38. ServidorPrincipal brisa = **new** ServidorPrincipal(puerto,index);
39. }
41. }

**Capturas de Pantalla.**

****

****

****

****

****

**Conclusión.**

En esta práctica so costo trabajo ya que el poder balancear las peticiones y las cargas fue mas difícil porque tuvimos que aplicar los conocimientos de las practicas anteriores y el problema fue hacerlo funcionar de manera conjunta sin que causara errores de petición o al detectar servidores y puertos ya ocupados cuando estábamos haciendo las practicas pero después de que logramos ejecutarlo de manera exitosa fue algo gratificante haber logrado esto.