Instituto Politécnico Nacional.

Escuela Superior De Cómputo.

Materia:

Aplicaciones Para Comunicaciones En Red.

Tema:

Reporte.

(Práctica 4).

Profesor:

Axel Ernesto Moreno Cervantes.

Alumno:

Luis Enrique Rojas Alvarado.

Mario Alberto Miranda Sandoval.

Grupo:

3CM5

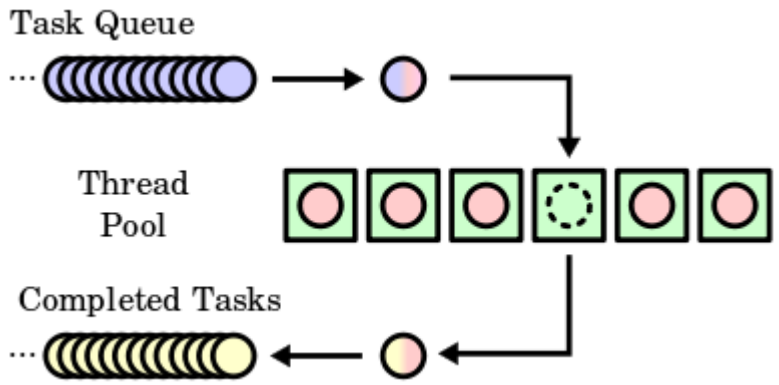
Introducción.

Pool de hilos.

En la programación de computadoras, un grupo de subprocesos es un patrón de diseño de software para lograr la concurrencia de ejecución en un programa de computadora. A menudo también llamado modelo de trabajadores replicados o modelo de trabajadores, un grupo de subprocesos mantiene varios subprocesos esperando que el programa de supervisión asigne

tareas para la ejecución concurrente. Al mantener un grupo de subprocesos, el modelo aumenta el rendimiento y evita la latencia en la ejecución debido a la frecuente creación y destrucción de subprocesos para tareas de corta duración. El número de subprocesos disponibles se ajusta a los recursos informáticos disponibles para el programa, como una cola de tareas paralelas después de la finalización de la ejecución.

El tamaño de un grupo de subprocesos es el número de subprocesos que se mantienen en reserva para ejecutar tareas. Suele ser un parámetro ajustable de la aplicación, ajustado para optimizar el rendimiento del programa. Decidir el tamaño óptimo del grupo de subprocesos es crucial para optimizar el rendimiento. Se ha sugerido la técnica de análisis de grupo de subprocesos (HTA) basada en la hipérbola para determinar el tamaño óptimo del grupo de subprocesos para el proceso de indexación basado en la nube en función de la carga de trabajo y el ancho de banda disponibles.



Mime.

Un tipo MIME es una etiqueta utilizada para identificar un tipo de datos. Se utiliza para que el software pueda saber cómo manejar los datos. Tiene el mismo propósito en Internet que las extensiones de archivo en Microsoft Windows.

Entonces, si un servidor dice "Esto es texto / html", el cliente puede ir "Ah, este es un documento HTML, puedo representarlo internamente", mientras que si el servidor dice "Esto es aplicación / pdf", el cliente puede ir "Ah", Necesito iniciar el complemento FoxIt PDF Reader que el usuario ha instalado y que se ha registrado como el controlador de aplicaciones / pdf.

Los encontrará más comúnmente en los encabezados de los mensajes HTTP (para describir el contenido con el que responde un servidor HTTP o el formato de los datos que se PUBLICAN en una solicitud) y en encabezados de correo electrónico (para describir el formato del mensaje y archivos adjuntos).

Desarrollo

En esta practica se hicieron peticiones como clientes a un servidor, que actúa como un apache, nos Brianda un servicio con el método POST y GET.

Como primer lugar veamos que son el método GET:

Cuando un usuario rellena un formulario en una página web los datos hay que enviarlos de alguna manera. Vamos a considerar las dos formas de envío de datos posibles: usando el método POST

Por ejemplo: <form action="http://www.aprenderaprogramar.com/prog/newuser" method ="get" >

En el ejemplo anterior la acción que se ejecutará cuando el usuario pulse el botón “Enviar” (submit) será el envío de los datos a la url especificada usando el método get.

Veamos el aspecto de un formulario cualquiera para hacernos una idea general.

Este formulario consta de varios campos que al usuario se le solicitan como Nombre, Apellidos, Correo electrónico, País y Mensaje. Posiblemente los nombres de los campos en el código HTML sean del tipo nombre_user, apellidos_user, email_user, pais_user y msg.

La diferencia entre los métodos get y post radica en la forma de enviar los datos a la página cuando se pulsa el botón “Enviar”. Mientras que el método GET envía los datos usando la URL, el método POST los envía de forma que no podemos verlos (en un segundo plano u "ocultos" al usuario).

Un resultado usando el método GET, a modo de ejemplo, podría ser el siguiente:

http://www.aprenderaprogramar.com/newuser.php?nombre=Pepe&apellido=Flores& email=h52turam%40uco.es&sexo=Mujer

El símbolo ? indica dónde empiezan los parámetros que se reciben desde el formulario que ha enviado los datos a la página.

Ahora veamos el método POST:

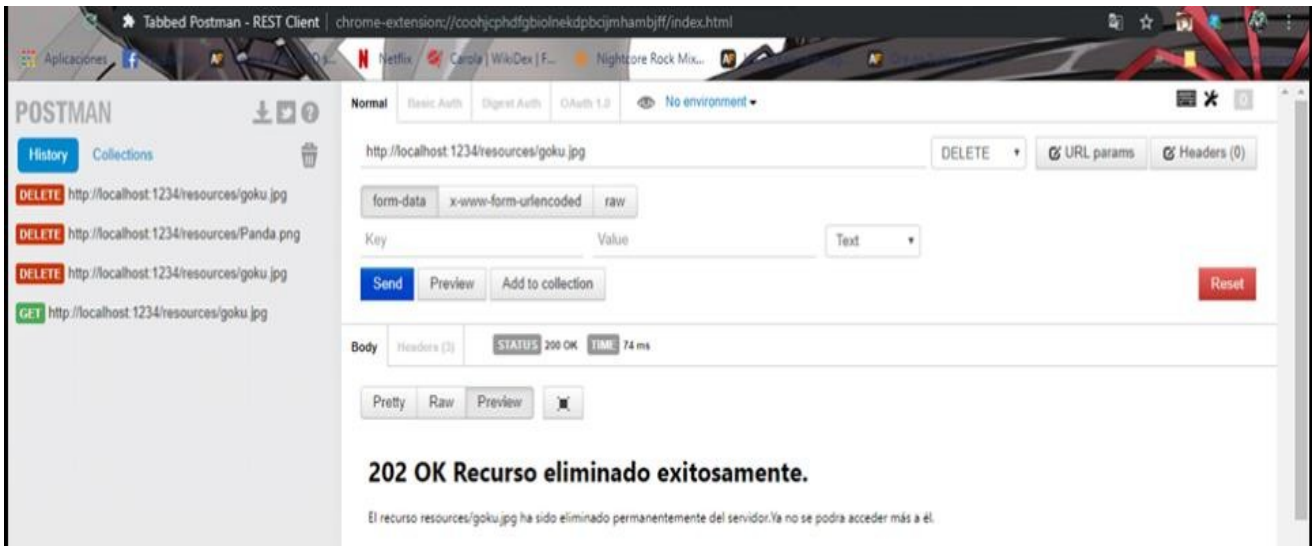
Hemos visto el resultado de un envío por el método GET. En el caso de un envío de datos usando el método POST, aunque estos datos también serán enviados (de una forma que podemos denominar “oculta”), no los podremos ver en la URL. Para poder recuperar los valores de los campos en el caso de un envío con el método POST necesitaríamos otras herramientas (por ejemplo, valernos del lenguaje PHP para recuperar el valor de esos campos).

El resultado final con ambos métodos podemos decir que es el mismo: la información se transmite de un lado a otro. La diferencia radica en que con el método GET podemos ver directamente los parámetros pasados ya que están dentro de la URL mientras que con el método POST los parámetros quedan ocultos y para rescatarlos hay que usar otras herramientas.

Un ejemplo de uso del método post sería este:

<form action="http://www.aprenderaprogramar.com/prog/newuser" method ="post">

Al hacer peticiones, necesitamos de un cliente, para poder ver mas a fondo de como funciona esta practica se uso POSTMAN que es una extensión del navegador Google Chrome, que permite el envío de peticiones HTTP REST sin necesidad de desarrollar un cliente.

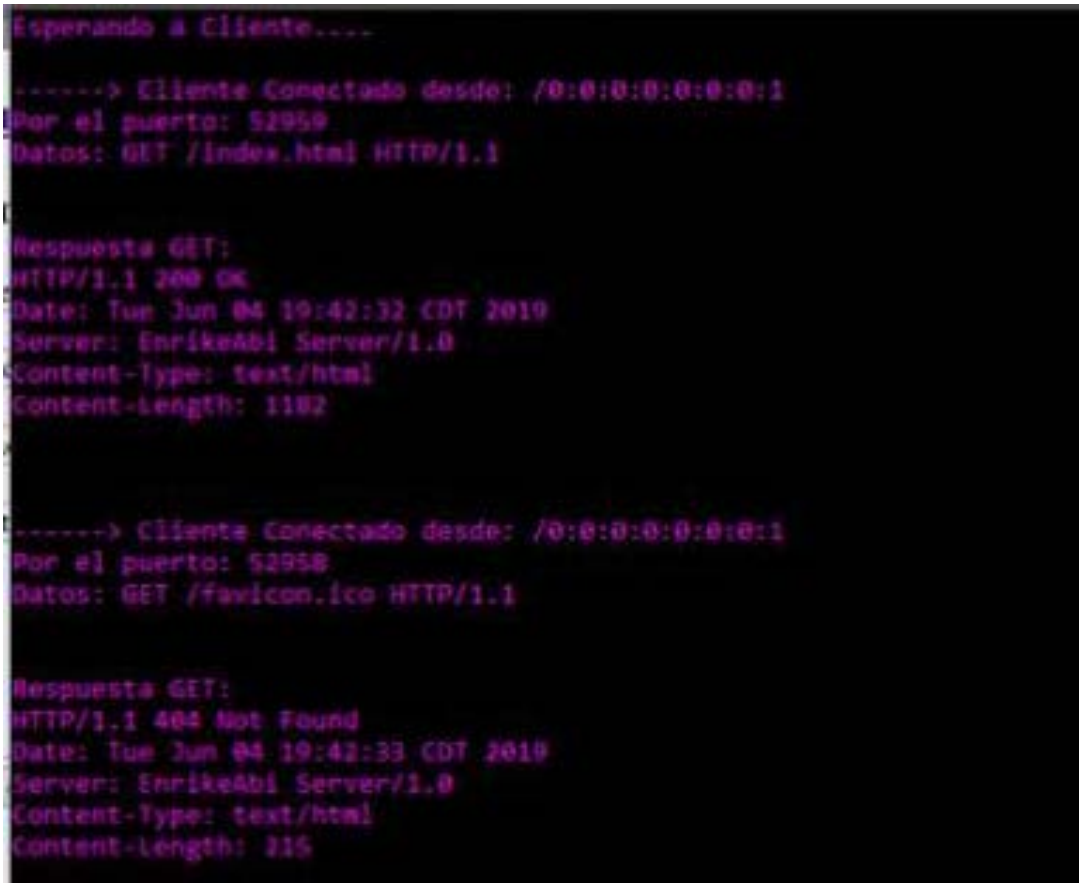


Y como primer lugar se hicieron dos formularios, uno que funciona con el método POST y otro que funciona con el método GET

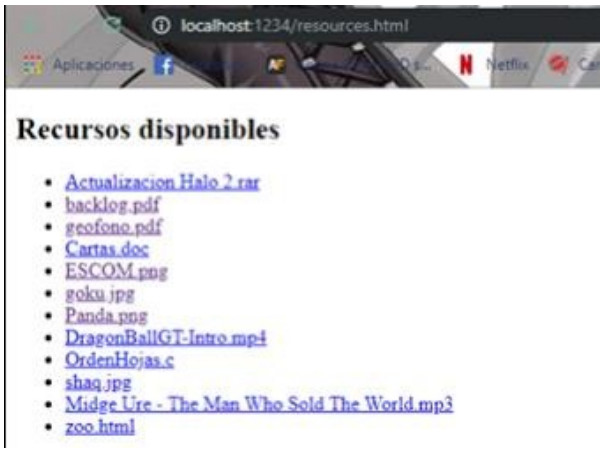
A continuación, se muestran los parámetros obtenidos por el método GET:

Y por el método POST:

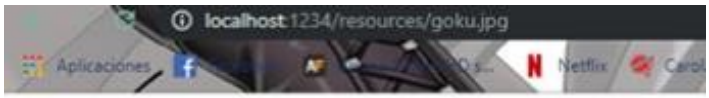
Por último, se muestran las peticiones de los clientes, estos se pueden observar cuando se corre el servidor, la siguiente captura muestra como cada cliente solicita consultas por método POST o GET:



Así mismo el servidor proporciona recursos disponibles como lo son imágenes, música, etc, que el usuario puede visualizar o acceder a ellos.



Y si el servidor borra uno de estos archivos y un cliente quiere acceder a él se muestra:



Error 404: Recurso no encontrado.

El sistema no puede encontrar el recurso especificado.

Código.

Manejador.java

```
1.  import java.net.*;
2.  import java.io.*;
3.  import java.util.*;
4.  import java.util.Base64;
5.
6.  public class Manejador extends Thread {
7.      protected Socket cl;
8.      protected DataOutputStream dos;
9.      protected Mime mime;
10.     protected DataInputStream dis;
11.
12.     public Manejador(Socket cl) throws Exception {
13.         this.cl = cl;
14.         this.dos = new DataOutputStream(this.cl.getOutputStream());
15.         this.mime = new Mime();
16.         this.dis = new DataInputStream(this.cl.getInputStream());
17.     }
18.
19.     public void eliminarRecurso(String arg, String headers){
20.         try {
21.             System.out.println(arg);
22.             File f = new File(arg);
23.
24.             if(f.exists()) {
25.                 if (f.delete()) {
26.                     System.out.println("------
    > Archivo " + arg + " eliminado exitosamente\n");
27.
28.                     String deleteOK = headers +
29.                                       "<html><head><meta charset='UTF-
    8'><title>202 OK Recurso eliminado</title></head>" +
30.                                       "<body><h1>202 OK Recurso eliminado exitosamente.</h1>"
    +
31.                                       "<p>El recurso " + arg + " ha sido eliminado permanentem
    ente del servidor." +
32.                                       "Ya no se podra acceder más a él.</p>" +
33.                                       "</body></html>";
34.
35.                     dos.write(deleteOK.getBytes());
36.                     dos.flush();
37.                     System.out.println("Respuesta DELETE: \n" + deleteOK);
38.                 }
39.                 else {
40.                     System.out.println("El archivo " + arg + " no pudo ser borrado\n");
41.
42.                     String error404 = "HTTP/1.1 404 Not Found\n" +
43.                                       "Date: " + new Date() + " \n" +
44.                                       "Server: EnrikeAbi Server/1.0 \n" +
45.                                       "Content-Type: text/html \n\n" +
46.
47.                                       "<html><head><meta charset='UTF-
    8'><title>404 Not found</title></head>" +
48.                                       "<body><h1>404 Not found</h1>" +
49.                                       "<p>Archivo " + arg + " no encontrado.</p>" +
50.                                       "</body></html>";
51.
52.                     dos.write(error404.getBytes());
```

```
                    dos.flush();
                    System.out.println("Respuesta DELETE - ERROR 404: \n" + error404);
                }
            }
        }
        catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public void enviarRecurso(String arg, int bandera) {
        try {
            File f = new File(arg);
            String sb = "HTTP/1.1 200 OK\n";

            if(!f.exists()) {
                arg = "404.html"; // Recurso no encontrado
                sb = "HTTP/1.1 404 Not Found\n";
            }
            else if(f.isDirectory()) {
                arg = "403.html"; // Recurso privado
                sb = "HTTP/1.1 403 Forbidden\n";
            }

            DataInputStream dis2 = new DataInputStream(new FileInputStream(arg));
            int tam = dis2.available();

            // Obtenemos extension para saber el tipo de recurso
            int pos = arg.indexOf(".");
            String extension = arg.substring(pos + 1, arg.length());

            // Enviamos las cabeceras de la respuesta HTTP - METODO HEAD
            sb = sb + "Date: " + new Date() + " \n" +
                      "Server: EnrikeAbi Server/1.0 \n" +
                      //Distintos tipos MIME para distintos tipos de archivos
                      "Content-Type: " + mime.get(extension) + " \n" +
                      "Content-Length: " + tam + " \n\n";

            dos.write(sb.getBytes());
            dos.flush();

            String metodo = "HEAD";
            if (bandera == 1) {
                metodo = "GET";
                // Respuesta GET, enviamos el archivo solicitado
                byte[] b = new byte[1024];
                long enviados = 0;
                       int n = 0;

                       while(enviados < tam) {
                           n = dis2.read(b);
                           dos.write(b, 0, n);
                           dos.flush();
                           enviados += n;
                       }
                   }
                   System.out.println("Respuesta " + metodo + ": \n" + sb);
                   dis2.close();
               }
               catch(Exception e) {
                   System.out.println(e.getMessage());
                   //e.printStackTrace();
               }
           }

           public String obtenerNombreRecurso(String line) {
               // Obtiene el nombre del recurso de la peticion HTTP
               int i = line.indexOf("/");
               int f = line.indexOf(" ", i);
               String resourceName = line.substring(i + 1, f);

               // Si es vacio, entonces se trata del index
               if(resourceName.compareTo("") == 0)
```

```
126.                     resourceName = "index.html";
127.
128.                 return resourceName;
129.             }
130.
131.             public String obtenerParametros(String line, String headers, int bandera) {
132.                 String metodo = "POST";
133.                 String request2 = line;
134.
135.                 if(bandera == 0) {
136.                     metodo = "GET";
137.                     // Line: GET /?Nombre=&Direccion=&Telefono=&Comentarios= HTTP/1.1
138.                     // Separamos los parametros de "GET"
139.                     System.out.println(line);
140.                     StringTokenizer tokens = new StringTokenizer(line, "?");
141.                     String request = tokens.nextToken();
142.                     request = tokens.nextToken();
143.
144.                     // Separamos los parametros de "HTTP/1.1"
145.                     StringTokenizer tokens2 = new StringTokenizer(request, " ");
146.                     request2 = tokens2.nextToken();
147.                 }
148.
149.                 System.out.println(request2);
150.                 // Separamos los parametros junto a su valor uno del otro
151.                 StringTokenizer paramsTokens = new StringTokenizer(request2, "&");
152.
153.                 String html = headers +
154.                                 "<html><head><meta charset='UTF-
    8'><title>Metodo " + metodo + "\n" +
155.                                 "</title></head><body bgcolor='#AACCFF'><center><h2>Parametros ob
    tenidos por medio de " + metodo + "</h2><br>\n" +
156.                                 "<table border='2'><tr><th>Parametro</th><th>Valor</th></tr>";
157.
158.                 // Se recorren todos los parametros, mientras existan
159.                 while(paramsTokens.hasMoreTokens()) {
160.                     String parametros = paramsTokens.nextToken();
161.                     // Separamos el nombre del parametro de su valor
162.                     StringTokenizer paramValue = new StringTokenizer(parametros, "=");
163.                     String param = ""; //Nombre del parametro
164.                     String value = ""; //Valor del parametro
165.
166.                     // Hay que revisar si existen o si se enviaron parametros vacios
167.                     if(paramValue.hasMoreTokens())
168.                         param = paramValue.nextToken();
169.
170.                     if(paramValue.hasMoreTokens())
171.                         value = paramValue.nextToken();
172.
173.                     html = html + "<tr><td><b>" + param + "</b></td><td>" + value + "</td></tr>
    \n";
174.                 }
175.                 html = html + "</table></center></body></html>";
176.                 return html;
177.             }
178.
179.             @Override
180.             public void run() {
181.                 // Cabeceras de respuestas HTTP
182.                 String headers = "HTTP/1.1 200 OK\n" +
183.                                  "Date: " + new Date() + " \n" +
184.                                  "Server: Wicho Server/1.0 \n" +
185.                                  "Content-Type: text/html \n\n";
186.                 try {
187.                     String line = dis.readLine(); // Lee primera linea DEPRECIADO !!!!
188.                     // Linea vacia
189.                     if(line == null) {
190.                         String vacia = "<html><head><title>Servidor WEB</title><body bgcolor='#
    AACCFF'>Linea Vacia</body></html>";
191.                         dos.write(vacia.getBytes());
192.                         dos.flush();
193.                     }
194.                     else {
```

```
195.                         System.out.println("\n------
    > Cliente Conectado desde: " + cl.getInetAddress());
196.                         System.out.println("Por el puerto: " + cl.getPort());
197.                         System.out.println("Datos: " + line + "\r\n\r\n");
198.
199.                         // Metodo GET
200.                         if(line.toUpperCase().startsWith("GET")) {
201.                             if(line.indexOf("?") == -1) {
202.                                 // Solicita un archivo
203.                                 String fileName = obtenerNombreRecurso(line);
204.                                 // Bandera HEAD = 0, GET = 1
205.                                 enviarRecurso(fileName, 1);
206.                             }
207.                             else {
208.                                 // Envia parametros desde un formulario
209.                                 // Bandera GET = 0, POST = 1
210.                                 String respuesta = obtenerParametros(line, headers, 0);
211.                                 // Respuesta GET, devolvemos un HTML con los parametros del for
    mulario
212.                                 dos.write(respuesta.getBytes());
213.                                 dos.flush();
214.                                 System.out.println("Respuesta GET: \n" + respuesta);
215.                             }
216.                         } // Metodo HEAD
217.                         else if(line.toUpperCase().startsWith("HEAD")) {
218.                             if(line.indexOf("?") == -1) {
219.                                 // Solicita archivo, unicamente enviamos tipo mime y longitud
220.                                 String fileName = obtenerNombreRecurso(line);
221.                                 // Bandera HEAD = 0, GET = 1
222.                                 enviarRecurso(fileName, 0);
223.                             }
224.                             else {
225.                                 // Respuesta HEAD, devolvemos unicamente las cabeceras HTTP
226.                                 dos.write(headers.getBytes());
227.                                 dos.flush();
228.                                 System.out.println("Respuesta HEAD: \n" + headers);
229.                             }
230.                         } // Metodo POST
231.                         else if(line.toUpperCase().startsWith("POST")) {
232.                             // Leemos el flujo de entrada
233.                             int tam = dis.available();
234.                             byte[] b = new byte[tam];
235.
236.                             dis.read(b);
237.                             //Creamos un string con los bytes leidos
238.                             String request = new String(b, 0, tam);
239.
240.                             // Separamos los parametros del resto de los encabezados HTTP
241.                             String[] reqLineas = request.split("\n");
242.                             //Ultima linea del request
243.                             int ult = reqLineas.length - 1;
244.
245.                             // Bandera GET = 0, POST = 1
246.                             String respuesta = obtenerParametros(reqLineas[ult], headers, 1);
247.
248.                             // Respuesta POST, devolvemos un HTML con los parametros del formul
    ario
249.                             dos.write(respuesta.getBytes());
250.                             dos.flush();
251.                             System.out.println("Respuesta POST: \n" + respuesta);
252.                         } // Metodo DELETE
253.                         else if(line.toUpperCase().startsWith("DELETE")) {
254.                             String fileName = obtenerNombreRecurso(line);
255.                             eliminarRecurso(fileName, headers);
256.                         }
257.                         else {
258.                             //Metodos no implementados en el servidor
259.                             String error501 = "HTTP/1.1 501 Not Implemented\n" +
260.                                               "Date: " + new Date() + " \n" +
261.                                               "Server: EnrikeAbi Server/1.0 \n" +
262.                                               "Content-Type: text/html \n\n" +
263.
```

```
                                                "<html><head><meta charset='UTF-
    8'><title>Error 501</title></head>" +
                                                "<body><h1>Error 501: No implementado.</h1>" +
                                                "<p>El método HTTP o funcionalidad solicitada no
    está implementada en el servidor.</p>" +
                                                "</body></html>";

                                dos.write(error501.getBytes());
                                dos.flush();
                                System.out.println("Respuesta ERROR 501: \n" + error501);
                            }
                        }
                        dis.close();
                        dos.close();
                        cl.close();
                    }
                    catch(Exception e) {
                        e.printStackTrace();
                    }
                }
            }
```

Mime.java

```
import java.util.*;

public class Mime {
    public static HashMap<String, String> mimeTypes;

    public Mime() {
        mimeTypes = new HashMap<>();
        mimeTypes.put("doc", "application/msword");
        mimeTypes.put("pdf", "application/pdf");
        mimeTypes.put("rar", "application/x-rar-compressed");
        mimeTypes.put("mp3", "audio/mpeg");
        mimeTypes.put("jpg", "image/jpeg");
        mimeTypes.put("jpeg", "image/jpeg");
        mimeTypes.put("png", "image/png");
        mimeTypes.put("html", "text/html");
        mimeTypes.put("htm", "text/html");
        mimeTypes.put("c", "text/plain");
        mimeTypes.put("txt", "text/plain");
        mimeTypes.put("java", "text/plain");
        mimeTypes.put("mp4", "video/mp4");
    }

    public String get(String extension) {
        if(mimeTypes.containsKey(extension))
            return mimeTypes.get(extension);
        else
            return "application/octet-stream";
    }

}
```

ServidorWeb.java

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ServidorWeb {

    public static void main(String[] args) {
        int pto, tamPool;

```

```
12.         try {
13.             Scanner sc = new Scanner(System.in);
14.             System.out.print("Puerto: ");
15.             pto = sc.nextInt();
16.             System.out.print("Tamanio del pool de conexiones: ");
17.             tamPool = sc.nextInt();
18.
19.             // Pool de Conexiones
20.             ExecutorService pool = Executors.newFixedThreadPool(tamPool);
21.             System.out.println("\n\n -----
    > Iniciando Servidor.... Pool de Conexiones = " + tamPool);
22.
23.             ServerSocket s = new ServerSocket(pto);
24.             System.out.println("Servidor iniciado: http://localhost:" + pto + "/ --- OK");
25.             System.out.println("Esperando a Cliente....");
26.
27.             for( ; ; ) {
28.                 Socket cl = s.accept();
29.                 Manejador manejador = new Manejador(cl);
30.                 pool.execute(manejador);
31.             }
32.         }
33.         catch(Exception e){
34.             e.printStackTrace();
35.         }
36.     }
37. }
```

Conclusiones.

Miranda Sandoval Mario Alberto.

En esta práctica se puede observar de manera perfecta el uso del pool de hilos, ya que este permite varias conexiones sin que este se vicie, además, se pudo observar el uso de los sockets normales, en esta ocasión para levantar un servidor.

Rojas Alvarado Luis Enrique

En esta práctica pudimos crear un servidor HTTP basado en el propio protocolo y hacer la réplica del servidor con los estándares mime para poder hacer la visualización de cualquier elemento que tengamos en nuestra computadora, como, por ejemplo: Un archivo con extensión mp3, o con extensión PDF.