## Práctica 7

1. **Alumno:** Alcausa Luque, Juan Carlos
2. **Titulación:** Grado en Ingeniería Informática + Matemáticas
3. **PC de la práctica:** portátil
4. **Ejercicio 1.** Explique cada una de las instrucciones enviadas por el cliente, indicando para qué se usa. ¿Cómo determina el servidor cuándo termina el cuerpo del correo?

* HELO RySD.2.0.24: para iniciar la sesión y crear la conexión
* MAIL FROM:<profesor@rysd.es>: indica el remitente del correo.
* RCPT TO:<alumnado@rysd.es>: indica uno de los destinatarios.
* RCPT TO:<profesorado@rysd.es>: indica otro destinatario.
* RCPT TO:<root@rysd.es>: indica el último de los destinatarios.
* DATA: se utiliza para indicar el inicio de la transferencia del cuerpo del mensaje de correo.
* Se envía el mensaje
* Se envía un QUIT para finalizar.

1. **Ejercicio 2.** ¿Por qué el servidor tras el comando DATA envía un código de tipo 3xx?

Porque indica que lo ha recibido correctamente y está a la espera de más datos (del contenido de correo)

1. **Ejercicio 3.** ¿Por qué hay tres envíos de comando RCPT TO?

Porque hay tres destinatarios en la traza que estamos analizando, uno es [alumnado@rysd.es](mailto:alumnado@rysd.es), otro [profesorado@rysd.es](mailto:profesorado@rysd.es) y el último [root@rysd.es](mailto:root@rysd.es)

**Ejercicio 4.** Si observa la imagen previa los destinatarios son enviados en los campos Para, CC (Carbon Copy) y BCC (Blind Carbon Copy). ¿Qué diferencia a nivel de comando SMTP y contenido del correo en sí mismo tiene que un destinatario sea indicado en un campo u otro?

En cuanto a comando todos son RCPT, que indica los receptores, es después de enviarse DATA cuando se indica de qué tipo queremos que sea cada uno.

**Escala de tiempo

Descripción generada automáticamente con confianza mediaEjercicio 5.** Use la opción **Follow TCP Stream** de Wireshark para observar el diálogo completo que han mantenido el cliente de correo y el servidor. Adjunte una captura de pantalla donde se observe dicho diálogo.

**Ejercicio 6.** ¿En cuales mensajes se usa *piggybacking*? ¿Por qué? En los que no usen esa estrategia, los mensajes de datos ¿confirman algo? ¿El qué?

Se usa en todos menos en los de datos, porque mientras se están enviando las líneas el servidor sólo responde con ACK, por lo que no hay nuevos datos que confirmar del servidor y el número de ACK permanece constante en estos mensajes en 88

**Ejercicio 7.** Si observa el interfaz gráfico de FakeSMTP, verá que este correo lo ha recibido varias veces. ¿Por qué?

Porque en la interfaz se nos muestra uno por cada destinatario.

**Ejercicio 8.** Sabría indicar (quizás mediante un uso “inteligente” del cliente desarrollado), si el servidor FakeSMTP es iterativo o concurrente. Justifique la respuesta y añada capturas de pantalla para apoyar su contestación.

Es un servidor concurrente ya que permite establecer varias conexiones a la vez. Si ejecutamos varias veces Main podemos tener varios procesos activos al mismo tiempo.

1. DESCRIPCIÓN DEL CÓDIGO:
2. **import** java.util.ArrayList;
3. **import** java.util.List;
4. **import** java.io.BufferedReader;
5. **import** java.io.IOException;
6. **import** java.io.InputStream;
7. **import** java.io.InputStreamReader;
8. **import** java.io.OutputStream;
9. **import** java.net.Socket;
10. **public** **class** Main {
12. // Socket para las comunicaciones
13. **static** Socket *socket* = **null**;
14. // Streams para el envío y recepción
15. **static** InputStream *in* = **null**;
16. **static** OutputStream *out* = **null**;
18. // Esta función se conecta al servidor de SMTP
19. **static** **void** conectar(){
20. **try** {
21. // **TODO**: conecte el socket al servidor de correo (máquina local en puerto 25)
22. *socket* = **new** Socket ("127.0.0.1", 25);
23. // Se abren los flujos de entrada y salida
24. *in* = *socket*.getInputStream();
25. *out* = *socket*.getOutputStream();
26. } **catch** (IOException e) {
27. System.***out***.println("Error al conectar al servidor: " + e.getMessage());
28. System.*exit*(-1);
29. }
30. }
32. // Esta función cierra el socket
33. **static** **void** desconectar(){
34. **try** {
35. // **TODO**: Cierre el socket y los flujos asociados
36. *socket*.close();
37. *in*.close();
38. *out*.close();
39. } **catch** (IOException e) {
40. System.***out***.println("Error al cerrar la conexión con el servidor: " + e.getMessage());
41. System.*exit*(-1);
42. }
43. }
45. // Envío de mensajes
46. **static** **void** enviar(String mensaje){
47. // Mostramos por consola el mensaje a enviar
48. System.***out***.println("C: " + mensaje);
50. **try** {
51. // **TODO**: Enviar usando write
52. // **TODO**: Recordar en añadir en cada mensaje el \r\n
53. mensaje = mensaje+"\r\n";
54. // **TODO**: Convertir a array de bytes con getBytes (no hace falta indicar el charset ni nada)
55. *out*.write(mensaje.getBytes());
56. // Usamos flush para forzar que el envio se haga en este momento
57. *out*.flush();
58. } **catch** (IOException e) {
59. System.***out***.println("Error al enviar: " + e.getMessage());
60. System.*exit*(-1);
61. }
62. }
64. // Recepción de mensajes
65. **static** **void** recibir(){
66. **byte** [] buffer = **new** **byte**[5000];
67. **try** {
68. // **TODO**: Recibir el mensaje con read
69. *in*.read(buffer);
70. } **catch** (IOException e) {
71. System.***out***.println("Error al recibir: " + e.getMessage());
72. System.*exit*(-1);
73. }
74. // Convertimos el mensaje recibido a String ...
75. String recv = **new** String(buffer,0,buffer.length);
76. // ... eliminamos el \r\n final ...
77. recv = recv.substring(0,recv.lastIndexOf('\r'));
78. // ... y lo mostramos por pantalla
79. System.***out***.println("S: "+ recv);
80. // **TODO**: Validar si el código obtenido es correcto (1xx, 2xx, 3xx) o incorrecto (4xx, 5xx)
81. **if**((**int**) recv.charAt(0) == 4 || (**int**) recv.charAt(0) == 5){
82. // **TODO**: Si es incorrecto enviar al servidor un RSET (use el método enviar)
83. *enviar*("RSET");
84. // **TODO**: Envíe luego el comando QUIT
85. *enviar* ("QUIT");
86. // Desconectamos del servidor
87. *desconectar*();
88. System.*exit*(0);
89. }
90. }
91. **public** **static** **void** main(String[] args) **throws** IOException {
92. // Flujo de lectura de teclado:
93. BufferedReader stdIn = **new** BufferedReader(**new** InputStreamReader(System.***in***));
95. // Nos conectamos al servidor
96. System.***out***.print("Conectado al servidor...");
97. *conectar*();
98. System.***out***.println("conectado!");
99. // ESQUEMA DEL PROTOCOLO SMTP
101. // Recibimos el saludo inicial del servidor
102. *recibir*();
104. // Enviamos el HELO y recibimos su respuesta
105. *enviar*("HELO servidor!");
106. *recibir*();
108. // Leemos el origen:
109. String origen = "";
110. **while**(origen.equals("")){
111. System.***out***.println("Dime el correo del emisor: ");
112. origen = stdIn.readLine();
113. }
114. // **TODO**: enviar origen del mensaje y recibir su respuesta
115. *enviar*("MAIL FROM: <" + origen + ">");
116. *recibir*();
118. // Ahora los destinos
119. List<String> destinos = **new** ArrayList<String>();
120. String destino = "";
121. **while**(destinos.size() == 0 || !destino.equals("")){
122. System.***out***.println("Dime el correo del destino (linea en blanco para acabar): ");
123. destino = stdIn.readLine();
124. **if**(!destino.equals("")){
125. destinos.add(destino);
126. // **TODO**: enviar el destino del mensaje y su recibir su respuesta
127. *enviar*("RCPT TO: <" + destino + ">");
128. *recibir*();
129. }
130. }
132. // Ahora enviamos el correo: cabeceras + cuerpo
133. // **TODO**: Enviamos el DATA y recibimos la respuesta
134. *enviar* ("DATA");
135. // Cabeceras:
136. // **TODO**: Enviar la cabecera From: (no hay que recibir respuesta)
137. *enviar* ("From: " + origen);
138. // **TODO**: Enviar las cabeceras To: (no hay que recibir respuesta)
139. **for** (String d : destinos) {
140. *enviar* ("To: " + d);
141. }
143. // Leemos el asunto:
144. String asunto = "";
145. **while**(asunto.equals("")){
146. System.***out***.println("Dime el asunto del correo: ");
147. asunto = stdIn.readLine();
148. }
149. // **TODO**: enviar la cabecera Subject: (no hay que recibir respuesta)
150. *enviar* ("Subject: " + asunto);
152. // Enviamos una línea en blanco para separar las cabeceras del cuerpo
153. *enviar*("");
155. // Ahora el cuerpo que son muchas líneas
156. String cuerpo = "";
157. System.***out***.println("Dime el mensaje (linea en blanco para acabar): ");
158. **do**{
159. cuerpo = stdIn.readLine();
160. **if**(!cuerpo.equals("")){
161. // **TODO**: enviar la línea (no hay que recibir respuesta)
162. *enviar*(cuerpo);
163. }
164. }**while**(!cuerpo.equals(""));
165. // **TODO**: Enviar una línea en blanco y luego un punto para acabar y recibir la respuesta
166. *enviar*("\r\n.");
167. *recibir*();
169. // **TODO**: Enviar QUIT para acabar y recibir la respuesta
170. *enviar* ("QUIT");
171. *recibir*();
173. // Nos desconectamos del servidor
174. System.***out***.print("Desconectándonos del servidor...");
175. *desconectar*();
176. System.***out***.println("Desconectado!");
177. }