

Memoria práctica 2.- Protocolos de Transporte

Sesión 1

- **Análisis del protocolo a emplear (SMTP RFC 5321)**

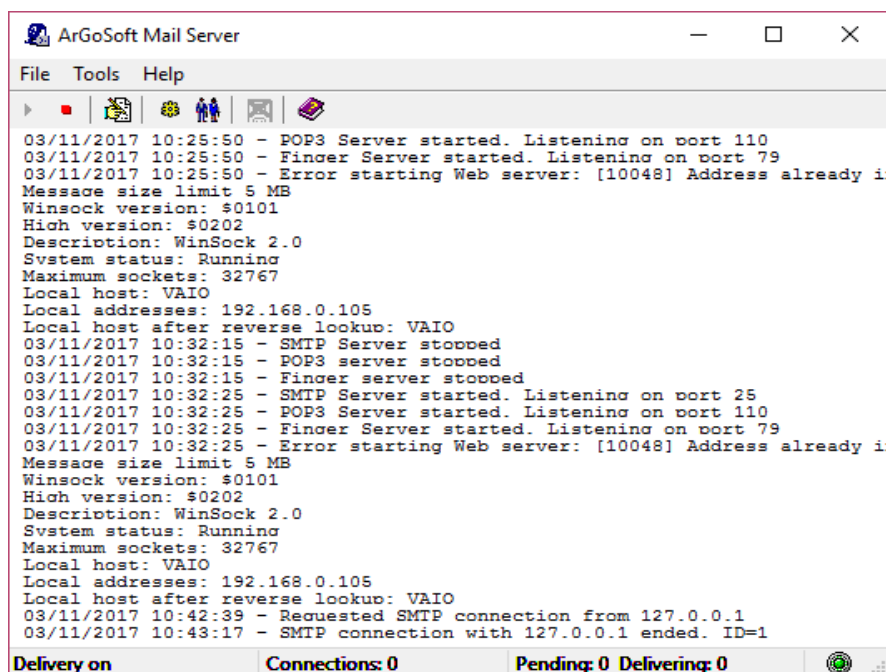
El **Simple Mail Transfer Protocol (SMTP)** o “protocolo para transferencia simple de correo”, es un protocolo de red utilizado para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos.

El funcionamiento de este protocolo se da en línea, de manera que opera en los servicios de correo electrónico. Sin embargo, este protocolo posee algunas limitaciones en cuanto a la recepción de mensajes en el servidor de destino (cola de mensajes recibidos). Como alternativa a esta limitación se asocia normalmente a este protocolo con otros, como el POP o IMAP, otorgando a SMTP la tarea específica de enviar correo, y recibirlos empleando los otros protocolos antes mencionados (POP O IMAP).

- **Prueba de su funcionalidad con un cliente genérico (TELNET)**

Cliente Telnet: El cliente Telnet permite a un equipo conectarse a un servidor Telnet remoto y ejecutar aplicaciones en dicho servidor. Una vez que el usuario ha iniciado una sesión, aparecerá el símbolo del sistema, que se puede utilizar como si se hubiera abierto localmente en la consola del servidor Telnet. Los comandos que se escriben en el símbolo del sistema del cliente Telnet se envían al servidor Telnet y se ejecutan allí, como si se hubiera iniciado una sesión localmente en el símbolo del sistema del servidor. Los resultados de los comandos ejecutados se devuelven al cliente Telnet, donde se muestran al usuario.

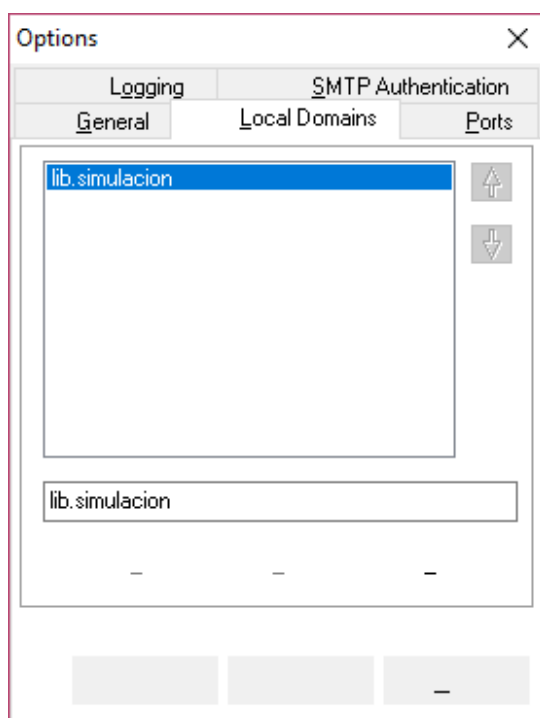
En primer lugar deberemos de crear un servidor de correo en nuestro ordenador con el software proporcionado por el profesor, como vemos en la siguiente imagen.



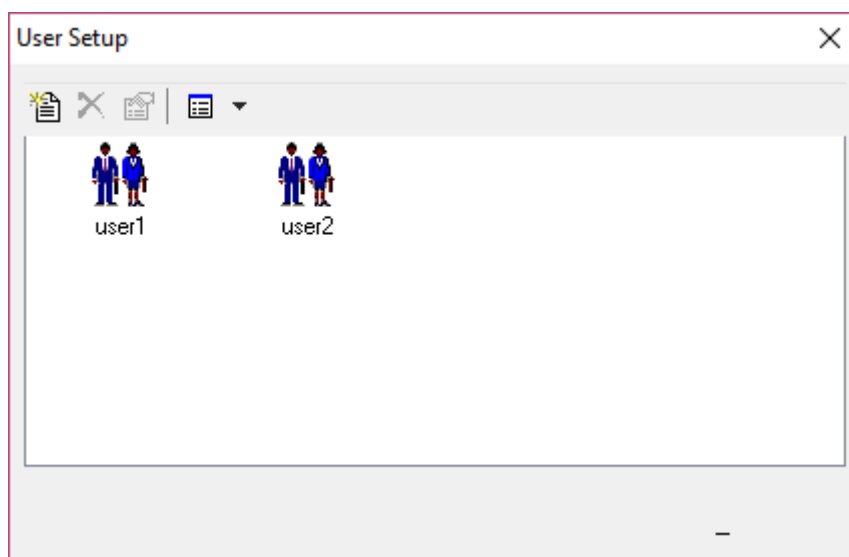
The screenshot shows the 'ArGoSoft Mail Server' application window. The title bar includes the application name and standard window controls. The menu bar has 'File', 'Tools', and 'Help'. The toolbar contains icons for running, stopping, and other server functions. The main text area displays a log of server activities, including the start and stop of POP3, SMTP, and Finger servers, and the status of the WinSock library. At the bottom, a status bar shows 'Delivery on', 'Connections: 0', and 'Pending: 0 Delivering: 0'.

```
ArGoSoft Mail Server
File Tools Help
03/11/2017 10:25:50 - POP3 Server started. Listening on port 110
03/11/2017 10:25:50 - Finger Server started. Listening on port 79
03/11/2017 10:25:50 - Error starting Web server: [10048] Address already in use
Message size limit 5 MB
Winsock version: $0101
High version: $0202
Description: WinSock 2.0
System status: Running
Maximum sockets: 32767
Local host: VAIO
Local addresses: 192.168.0.105
Local host after reverse lookup: VAIO
03/11/2017 10:32:15 - SMTP Server stopped
03/11/2017 10:32:15 - POP3 server stopped
03/11/2017 10:32:15 - Finger server stopped
03/11/2017 10:32:25 - SMTP Server started. Listening on port 25
03/11/2017 10:32:25 - POP3 Server started. Listening on port 110
03/11/2017 10:32:25 - Finger Server started. Listening on port 79
03/11/2017 10:32:25 - Error starting Web server: [10048] Address already in use
Message size limit 5 MB
Winsock version: $0101
High version: $0202
Description: WinSock 2.0
System status: Running
Maximum sockets: 32767
Local host: VAIO
Local addresses: 192.168.0.105
Local host after reverse lookup: VAIO
03/11/2017 10:42:39 - Requested SMTP connection from 127.0.0.1
03/11/2017 10:43:17 - SMTP connection with 127.0.0.1 ended. ID=1
Delivery on Connections: 0 Pending: 0 Delivering: 0
```

Introducimos el dominio.

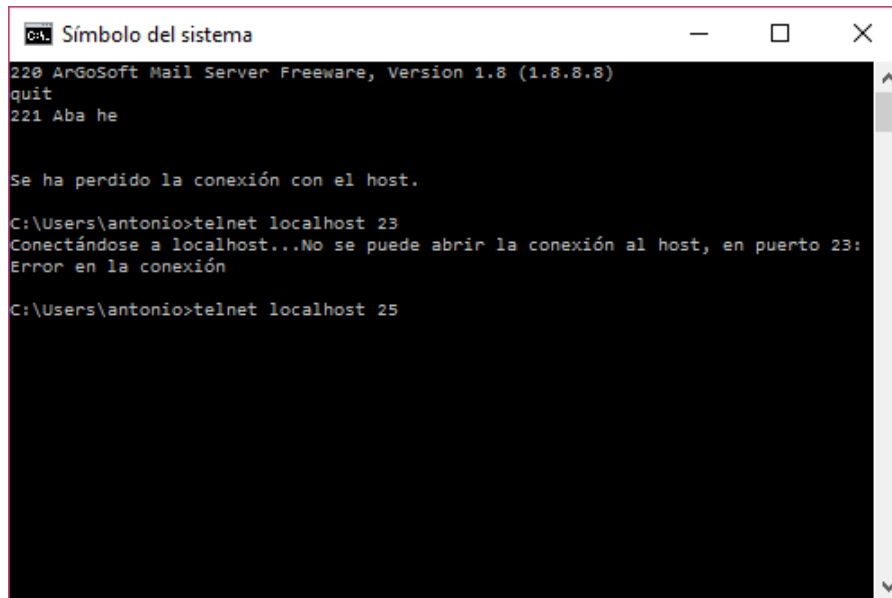


Introducimos los usuarios que podrán comunicarse en la red:



A continuación, iniciamos el cliente Telnet utilizando el servidor de correo creado anteriormente. Como el servidor de correo se encuentra en nuestro ordenador, deberemos introducir “**localhost**” junto al puerto por defecto.

El puerto por defecto de Telnet es 23, mientras que el puerto por defecto de SMTP es 25. El puerto por defecto de POP3 es 110.



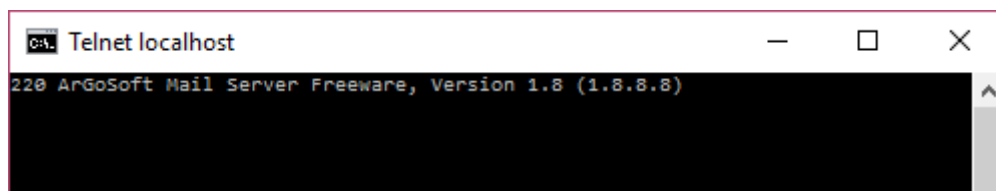
```
220 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)
quit
221 Aba he

Se ha perdido la conexión con el host.

C:\Users\antonio>telnet localhost 23
Conectándose a localhost...No se puede abrir la conexión al host, en puerto 23:
Error en la conexión

C:\Users\antonio>telnet localhost 25
```

Introduciendo el puerto por defecto de SMTP, nos permite acceder al cliente Telnet.

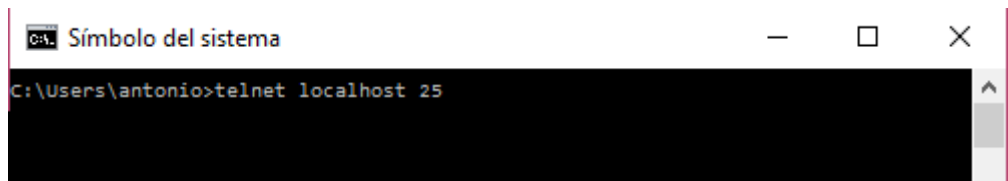


```
Telnet localhost

220 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)
```

Realizaremos un ejemplo de envío de un correo mediante SMTP y lo recibimos mediante POP3.

Utilizamos el cliente telnet para conectarnos al servidor de correo SMTP (puerto 25).



Comandos SMTP

NEMÓNICO	SINTAXIS ¹	DESCRIPCIÓN
HELO	"HELO" SP Domain CRLF	Identifica remitente, actualmente solamente se usa si no se requiere la funcionalidad extendida del servidor. Se mantiene por compatibilidad. Los clientes deben usar siempre EHLO (ver a continuación)
EHLO	"EHLO" SP (Domain / address-literal) CRLF	Permite usar las extensiones de SMTP definidas a partir de la RFC 1425 y subsecuentes estándares. Todos los servidores deben soportar este comando aunque no implementen ninguna extensión. <i>El dominio debe ser un nombre primario del host como el que se obtiene de una petición a un registro de recursos de DNS o una dirección IP literal si no dispone de dominio.</i>
MAIL	"MAIL FROM:" Reverse-path [SP <mail-parameters>] CRLF	Comienza la transacción de correo e identifica al remitente
RCPT	"RCPT TO:" ("<Postmaster@" Domain ">" / "<Postmaster>" / Forward-path) [SP Rcpt-parameters] CRLF	Identifica al destinatario. Pueden existir múltiples comandos <i>RCPT</i> , permitiendo el envío del mismo correo a los destinatarios indicados en el comando
DATA	"DATA" CRLF	Indica que el remitente está listo para transmitir una serie de líneas de texto, cada una finalizada CRLF. Una línea que únicamente contiene .CRLF indica el fin de datos
QUIT	"QUIT" CRLF	Finaliza la sesión SMTP
RSET	"RSET" CRLF	Aborta la transacción en curso y reinicia la sesión
VRFY	"VRFY" SP String CRLF	(Verify) Confirma que el nombre es un destinatario válido
EXPN	"EXPN" SP String CRLF	Lista los componentes de una lista de correo
NOOP	"NOOP" [SP String] CRLF	Responde con un código de asentimiento positivo (250 OK)
HELP	"HELP" [SP String] CRLF	Muestra ayuda sobre el servidor o sobre más específicamente sobre lo solicitado en la cadena si fuera posible (esta opción no es obligatoria)

```
Telnet localhost
220 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)
helo @lib.simulacion
250 Welcome [127.0.0.1], pleased to meet you
MAIL from: user1@lib.simulacion
250 Sender "user1@lib.simulacion" OK...
RCPT to: user2@lib.simulacion
250 Recipient "user2@lib.simulacion" OK...
DATA
354 Enter mail, end with "." on a line by itself
Hola, esto es un mensaje enviado mediante SMTP
.
250 Message accepted for delivery. <t8h7kufbhouzqx.031120171201@VAIO>
```

A continuación, utilizamos el cliente Telnet para conectarnos al servidor de correo POP3 (puerto 110), para poder mostrar nuestro mensaje.

```
Símbolo del sistema
C:\Users\antonio>telnet localhost 110
```

Comandos POP3

NEMÓNICO	DESCRIPCIÓN	RESTRICCIONES
USER <user>	Especifica un nombre de usuario	Solamente en el estado de AUTORIZACIÓN o tras un USER o PASS infructuoso.
PASS <pass>	Especifica contraseña	Solamente en el estado de AUTORIZACIÓN y tras un USER con éxito.
STAT	Estado del almacén (bandeja) de entrada: número total de mensajes y el total de bytes ocupados.	Solamente en el estado de TRANSACCIÓN
LIST [<M>]	Proporciona la lista de los mensajes y el tamaño de todos si no se aporta el parámetro M, uno por línea. La última línea contiene '.', Si no, se lista el tamaño de solo el mensaje M	Solamente en el estado de TRANSACCIÓN
RETR <M>	Recupera un mensaje	Solamente en el estado de TRANSACCIÓN
DELE <M>	Marca un mensaje para ser borrado del almacén (bandeja) de entrada	Solamente en el estado de TRANSACCIÓN
NOOP	Devuelve un asentimiento positivo (+ok)	Solamente en el estado de TRANSACCIÓN
RSET	Todas las marcas de borrado se desactivan	Solamente en el estado de TRANSACCIÓN
QUIT	Borra los mensajes marcados y devuelve el resultado de esta operación. Además cierra la conexión TCP	Ninguna

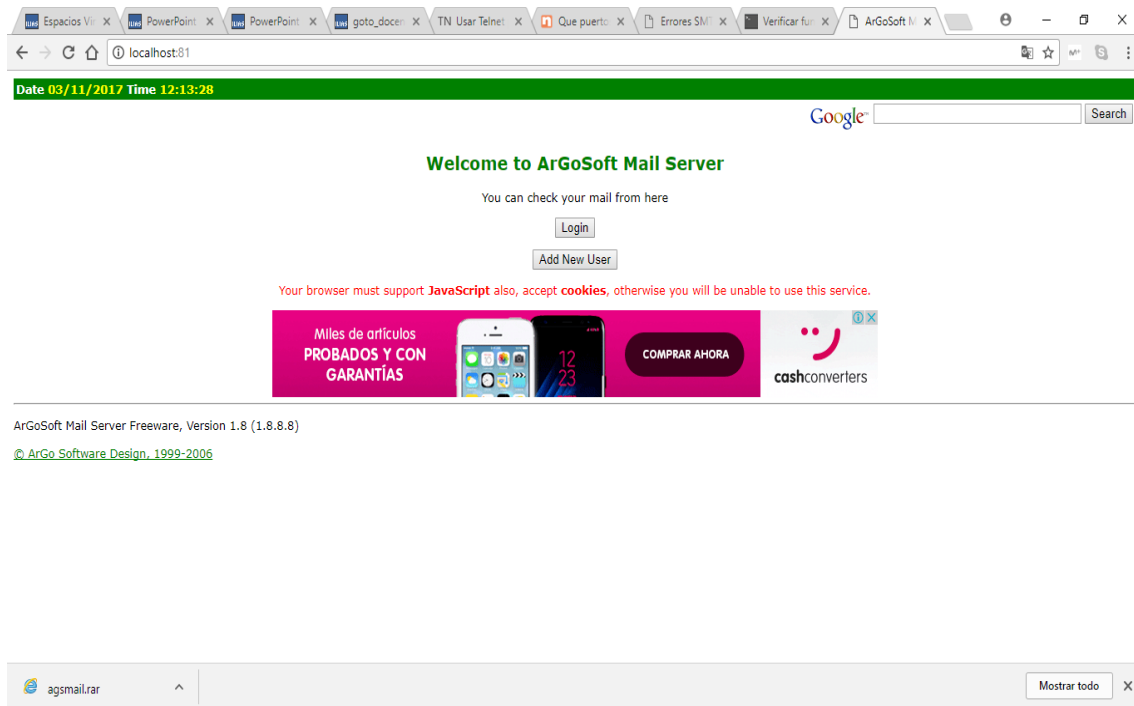
```
Telnet localhost
+OK ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)
USER user2
+OK Password required for user2
PASS Usuario2
+OK Mailbox locked and ready
LIST
+OK
1 239
2 265
.
RETR 2
+OK 265 octets
Received: from [127.0.0.1] by VAIO
(ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)); Fri, 3 Nov 2017 12:01:
03 +0100
Message-ID: <t8h7kufbhouzqx.031120171201@VAIO>
Date: Fri, 3 Nov 2017 12:01:03 +0100

Hola, esto es un mensaje enviado mediante SMTP
.
```

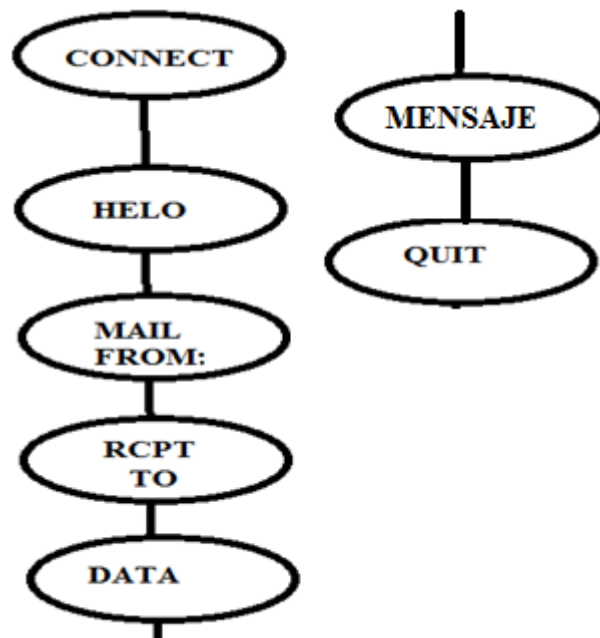
Nota:

Mirando que puerto tiene la WEB INTERFACE, e introduciendo en navegador google
→ “localhost:81” podemos usar una interfaz web en vez de emplear la consola con sus
respectivos comandos.

Options	
SMTP Authentication	
General	Local Domains
SMTP	25 Default
POP3	110 Default
Finger	79 Default
Web Interface	81 Default



- **Planificación de la estructura del cliente: diseño de la máquina de estados.**



SESION 2

- **Primeras modificaciones en el cliente básico TCP de la práctica 1.**

Añadir soporte a los comandos básicos: HELO y QUIT.

Establecer un control de errores básico.

Lo primero que debemos hacer es definir los comandos HELO y QUIT en el archivo cabecera (protocolo.h).

```
1  #ifndef protocolostpte_practicas_headerfile
2  #define protocolostpte_practicas_headerfile
3  #endif
4
5  // COMANDOS DE APLICACION
6  //Definimos comando HELO
7  #define HE "HELO"
8  //Definición del comando QUIT
9  #define SD "QUIT" // Finalización de la conexión de aplicación
10
11 #define SC "USER" // SOLICITUD DE CONEXION USER usuario
12 #define PW "PASS" // Password del usuario PASS password
13 #define SD2 "EXIT" // Finalización de la conexión de aplicación
14 #define ECHO "ECHO" // Definición del comando "ECHO" para el servicio de eco
15
```

A continuación, definimos la máquina de estados para los comandos anteriormente introducidos.

```
13 #define SUZ "EXIT" // Finalización de la conexión de aplicación
14 #define ECHO "ECHO" // Definición del comando "ECHO" para el servicio de eco
15
16
17 // RESPUESTAS A COMANDOS DE APLICACION
18 #define OK "OK"
19 #define ER "ER"
20
21 //FIN DE RESPUESTA
22 #define CRLF "\r\n"
23
24 //ESTADOS
25 #define S_HELO 0
26 #define S_USER 1
27 #define S_PASS 2
28 #define S_DATA 3
29 #define S_QUIT 4
30 #define S_EXIT 5
```

Y definimos el puerto de SMTP. Dicho puerto es el 25.

```
47  /*
48  //PUERTO DEL SERVICIO
49  #define TCP_SERVICE_PORT 6000
50  */
51  //Introducimos el puerto por defecto de SMTP
52  #define default_mailPort 25
53
54  /*No necesarios NOMBRE Y PASSWORD en esta practica
55  #define USER "alumno"
56  #define PASSWORD "123456"
57  */
```


Declaramos el estado S_HELO y S_QUIT

```
114         if(connect(sockfd, server_in, address_size)==0){
115             printf("CLIENTE> CONEXION ESTABLECIDA CON %s:%d\r\n",ipdest, default_mailPort);
116             recibidos = recv(sockfd, buffer_in, 512, 0); //recibimos la informacion del servidor para saber que esta listo
117             //Inicio de la máquina de estados
118             do {
119                 switch (estado) {
120                     case S_HELO:
121                         // Se recibe el mensaje de bienvenida
122                         printf("\nBienvenido a SEVICEMAIL-ANFE\r\n");
123                         sprintf_s(buffer_out, sizeof(buffer_out), "HELO %s %s", ipdest, CRLF); //250 correcto
124                         estado++;
125                         break;
126
127                     case S_QUIT:
128                         break;
129                 }
130             } while (1);
131
132             if(estado!=S_HELO){
133                 enviados=send(sockfd,buffer_out,(int)strlen(buffer_out),0);
134                 if(enviados==SOCKET_ERROR){
135                     estado=S_QUIT;
136                     continue;
137                 }
138             }
139         }
```

En la práctica nos pide que introduzcamos la fecha a la que se envía el mensaje, veamos cómo se haría:

Añadimos librería

```
//Libreria para tiempo
#include <time.h>
```

Variables utilizadas:

```
57 //Variables para la fecha
58 time_t tiempo = time(0);
59 struct tm *tlocal = localtime(&tiempo);
60 char salida_tiempo[128];
61 strftime(salida_tiempo, 128, "%d/%m/%y %H:%M",tlocal);
62
```

A continuación, lo introducimos en el estado S_MENSAJE como veremos en el siguiente apartado.

Tercera sesión:

Añadir los comandos de envío de mensajes.

Completar el control de errores del protocolo de aplicación.

Añadir funciones de interfaz de usuario básicas, como salida, reconexión y comunicación de errores al usuario.

Definimos los comandos de envío de mensajes en protocolo.h .

```
5  // COMANDOS DE APLICACION
6  //Definimos comando HELO
7  #define HE "HELO"
8  //Definimos los comandos para envío de mensaje
9  //MAIL FROM
10 #define MA "MAIL FROM: "
11 //RCPT TO
12 #define RCPT "RCPT TO: "
13 //DATA
14 #define DATA "DATA"
15 //MENSAJES
16 #define MENS "MENS"
17
18
19 #define SC "USER" // SOLICITUD DE CONEXION USER usuario
20 #define PW "PASS" // Password del usuario PASS password
21 #define SD2 "EXIT" // Finalizacion de la conexion de aplicacion
22 #define ECHO "ECHO" // Definicion del comando "ECHO" para el servicio de eco
23 //Definición del comando QUIT
24 #define SD "QUIT" // Finalizacion de la conexion de aplicacion
```

Definimos la máquina de estados para el envío de mensajes

```
35 //ESTADOS
36 //Definimos los estados para el envío de mensajes
37 #define S_HELO 0
38 #define S_MAIL 1
39 #define S_RCPT 2
40 #define S_DATA 3
41 #define S_MENSAJE 4
42 #define S_QUIT 5
43 #define S_EXIT 6
44
45
46 //No necesarios por ahora
47 #define S_VRFY 8
48 #define S_NOOP 9 //Responde codigo asentamiento positivo 250 ok
49 #define S_HELP 10
```

Declaramos los estados S_MAIL, S_RCPT, S_DATA, S_MENSAJE en la máquina de estados:

```
173 case S_MAIL:
174     printf("MAIL FROM: (enter para salir):");
175     //Lee los caracteres de la entrada estándar y los almacena como una
176     //cadena hasta que se alcanza un carácter de nueva línea
177     gets_s(input, sizeof(input));
178     if (strlen(input) == 0) {
179         // Si la longitud de input es 0, pasamos al estado QUIT
180         sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", SD, CRLF); //SD -> QUIT
181         estado = S_QUIT;
182     }
183     else {
184         //Escribimos MAIL FROM: input y crlf y pasamos al siguiente estado
185         sprintf_s(buffer_out, sizeof(buffer_out), "%s %s%s", MA, input, CRLF);
186
187         estado++;
188     }
189     break;
```

```
148 case S_RCPT:
149     printf("RCPT TO: ");
150     gets_s(input2, sizeof(input2));
151
152     if (strlen(input2) == 0) {
153         // Si la longitud de input es 0, pasamos al estado QUIT
154         sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", SD, CRLF); //SD -> QUIT
155         estado = S_QUIT;
156     }
157     else {
158         //Escribimos RCPT TO: input y crlf y pasamos al siguiente estado
159         sprintf_s(buffer_out, sizeof(buffer_out), "%s%s%s", RCPT, input2, CRLF);
160
161         estado++;
162     }
163     break;
```

```

209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
case S_DATA:
    printf("CLIENTE> Introduzca datos (enter para salir): ");
    //Introducimos datos
    gets_s(input4, sizeof(input4));
    //Si el tamaño de input4 es 0, nos salimos
    if (strlen(input4) == 0) {
        sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", SD, CRLF);
        estado = S_QUIT;
    }
    else {
        //Escribimos input y pasamos al siguiente estado
        printf("DATA: %s%s", input4, CRLF);
        estado++;
    }
    break;

```

```

225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
case S_MENSAJE:
    //Escribimos asunto
    printf("Asunto:");
    gets(input3);
    printf("\nMensaje de correo: \r\n");
    printf("Fecha y hora: %s \r\n", salida_tiempo); //Nos da la hora en la que se ha enviado
    printf("Asunto: %s%s", input3, CRLF);
    printf("Remitente: %s%s%s", MA, input, CRLF);
    printf("Destinatario: %s%s%s", RCPT, input2, CRLF);
    printf("Datos: %s%s", input4, CRLF);
    //Enviamos mensaje
    enviados = send(sockfd, buffer_out, (int)strlen(buffer_out), 0);
    printf("SERVIDOR> Datos enviados correctamente\r\n");
    estado++;
    break;

```

Funciones de interfaz de usuario básicas, como salida, reconexión y comunicación de errores al usuario.

Para la salida podemos ver que en los distintos estados implementamos bucles **if** que permiten la salida en caso de no querer enviar correo.

Para el caso de reconexión en la siguiente sesión estableceremos un estado RSET que permitirá una reconexión para volver a escribir un correo sin tener que salirte del servicio.

A continuación, mostramos un control de errores. Este control de errores permitirá, en caso de introducir un destinatario inexistente en el servidor, volver a introducir dicho destinatario o incluso, volver a introducir tanto el usuario que envía el mensaje como el que recibe dicho mensaje.

Veamos cómo sería en código:

```
270     recibidos=recv(sockfd,buffer_in,512,0);
271     if(recibidos<=0){
272         DWORD error=GetLastError();
273         if(recibidos<0){
274             printf("CLIENTE> Error %d en la recepción de datos\r\n",error);
275             estado=S_QUIT;
276         }
277         else{
278             printf("CLIENTE> Conexión con el servidor cerrada\r\n");
279             estado=S_QUIT;
280         }
281     }
282     else {
283         buffer_in[recibidos] = 0x00;
284         //Nos permite escribir un mensaje de envío correcto y recepción correcta
285         //Ponemos los estados S_RCPT y S_DATA debido a que queremos que escriba el mensaje
286         //en el estado mail , rcpt y helo --> en helo estado++ (mail),en mail estado++(RCPT) y rcpt estado++(DATA)
287         if (estado == S_MAIL || estado == S_RCPT || estado == S_DATA) {
288             //Escribe un mensaje de envío correcto y recepción correcta
289             printf(buffer_in);
290             //Definimos en protocol.h una respuesta a un comando de aplicacion
291             //Será UU y significa: "554 User unknown"
292             //Si el buffer_in es usuario erroneo
293             if (strcmp(buffer_in, UU, 2) == 0) {
294                 //Definimos variable para introducir enteros
295                 int estado2=0;
296                 do {
297                     printf("Introduce 1 --> Introducir los dos usuarios de nuevo\n2--> Para introducir un usuario correcto\n");
298                     //scanf_s necesita el tipo de dato %i y direccion &.
299                     scanf_s("%i", &estado2);
300                     switch (estado2) {
301                         case 1:
302                             estado = S_MAIL;
303                             //Nos permite poder escribir de nuevo en el case S_MAIL
304                             gets_s(input, sizeof(input));
305                             break;
306                         case 2:
307                             estado = S_RCPT;
308                             gets_s(input2, sizeof(input2));
309                             break;
310                         default:
311                             printf("Opcion no disponible\n");
312                             break;
313                     }
314                 } while (estado2 != 1 && estado2 != 2);
315             }
316         }
317     }
318 }
319 }
320 }
321 }while(estado!=S_QUIT);
```

Nota:

Cuando introducíamos un usuario incorrecto, el servidor nos devolvía “554 User unknown”, por lo tanto, para poder controlarlo, simplemente añadimos, en protocol.h, una respuesta a un comando de aplicación.

```
25 // RESPUESTAS A COMANDOS DE APLICACION
26 #define OK "OK"
27 #define ER "ER"
28 #define OKDATA "OD"
29 #define UU "554 User unknown"
30
```

Cuarta sesión:

Redacción de correos de cualquier longitud.

Soporte al comando RSET.

Para poder introducir correo de mayor longitud basta con definir un vector input[] de mayor tamaño.

A continuación, mostramos como dar soporte a RSET.

Definimos el comando de aplicación en protocol.h

```
5  // COMANDOS DE APLICACION
6  //Definimos comando HELO
7  #define HE "HELO"
8  //Definimos los comandos para envío de mensaje
9  //MAIL FROM
10 #define MA "MAIL FROM: "
11 //RCPT TO
12 #define RCPT "RCPT TO: "
13 //DATA
14 #define DATA "DATA"
15 //MENSAJES
16 #define MENS "MENSAJE"
17 //SESION 4
18 //RSET: Aborta la transacción en curso y reinicia la sesión
19 #define RSET "RSET"
--
```

Y definimos el estado S_RSET

```
34 //ESTADOS
35 //Definimos los estados para el envío de mensajes
36 #define S_HELO 0
37 #define S_MAIL 1
38 #define S_RCPT 2
39 #define S_DATA 3
40 #define S_MENSAJE 4
41 #define S_RSET 5
42 #define S_QUIT 6
43 #define S_EXIT 7
..
```

NOTA:

El estado S_RSET debe de ir antes que el estado S_QUIT, ya que S_QUIT se encarga de finalizar la conexión.

Declaramos el estado S_RSET en la máquina de estados:

```
190         case S_RSET:
191             do {
192                 printf("¿Desea escribir otro mensaje? (s/n)\r\n");
193                 character = _getche(); //Lee character
194             } while (character!= 's' && character!= 'n' && character!= 'S' && character!= 'N');
195             if (character == 'S' || character == 's') {
196                 estado = S_HELO;
197             }
198             else {
199                 estado++;
200             }
201             break;
202
```

Para redacción de correos de cualquier longitud.

No hemos encontrado la forma de hacerlo, no obstante, hemos definido un buffer de salida de tamaño un poco más grande

```
32 int main(int *argc, char *argv[])
33 {
34     //Definición de variables
35     SOCKET sockfd; //Crea el socket
36     //Estructura host (SESION 5)
37     struct hostent *host;
38     struct in_addr address; //SESION 5
39     struct sockaddr *server_in;
40     struct sockaddr_in server_in4;
41     struct sockaddr_in6 server_in6;
42     int address_size = sizeof(server_in4);
43     char buffer_in[1024], input[1024], input2[1024], input3[1024], input4[1024];
44     char buffer_out[6144]; //Buffer de salida le pondremos un tamaño grande.
45     int recibidos=0, enviados=0;
46     int estado=S_HELO;
47     char *opcion;
```

Quinta sesión:

Pruebas.

Añadir la resolución de dominios.

Para añadir la resolución a dominios hemos utilizado el código proporcionado por el profesor, haciendo un **strcpy** adicional, simplemente para copiar en **ipdest** la dirección de **address**

```
103 printf_s("CLIENTE> Socket CREADO\n");
104 printf("CLIENTE> Introduzca la IP destino (pulsar enter para IP por defecto) o dominio: ");
105 gets_s(ipdest,sizeof(ipdest));
106 //SESION 5
107 //Codigo profesor diapositivas(No introducir dominio de argosoft)
108 ipdestl = inet_addr(ipdest);
109 if (ipdestl == INADDR_NONE) {
110     //La dirección introducida por teclado no es correcta o
111     //corresponde con un dominio.
112     struct hostent *host;
113     host = gethostbyname(ipdest); //Pruebo si es dominio
114     if (host != NULL) { //Si es distinto de null, es dominio
115         memcpy(&address, host->h_addr_list[0], 4); //Tomo los 4 primeros bytes.
116         printf("\nDireccion %s\n", inet_ntoa(address));
117     }
118     //Copia en ipdest
119     strcpy_s(ipdest,sizeof(ipdest),inet_ntoa(address));
120 }
121
```

Protocolos de transporte

Antonio Osuna Melgarejo

Fernando Cabrera Caballero

3ºCurso

Ingeniería Telemática

