

Memoria práctica 2.- Protocolos de Transporte

Sesión 1

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

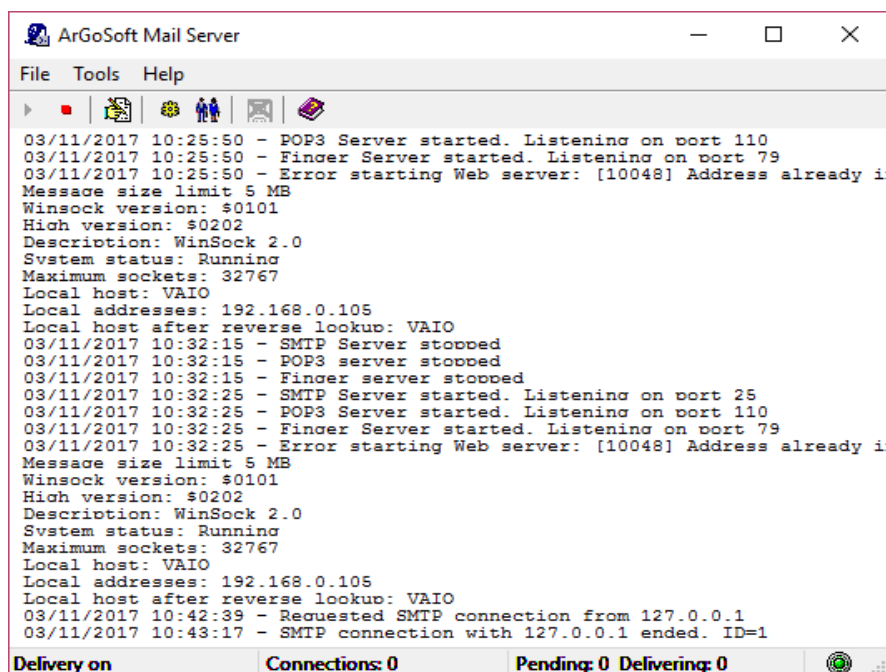
El **Simple Mail Transfer Protocol (SMTP)** o “protocolo para transferencia simple de correo”, es un protocolo de red utilizado para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos.

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).

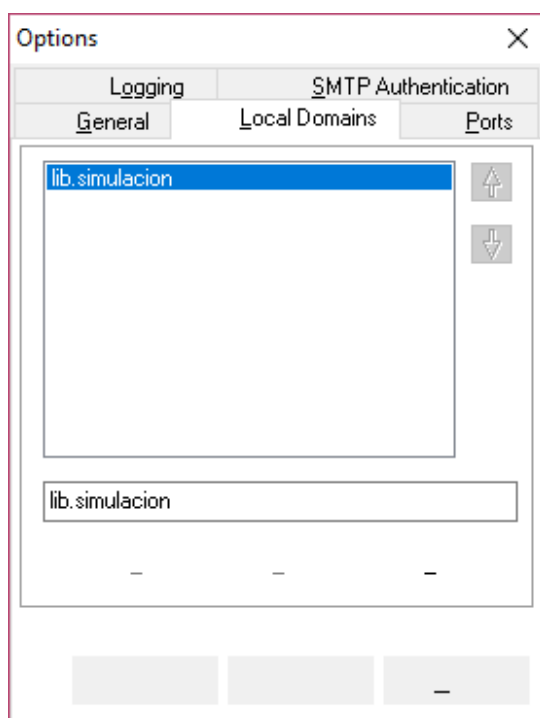
- **Prueba de su funcionalidad con un cliente genérico (TELNET)**

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.

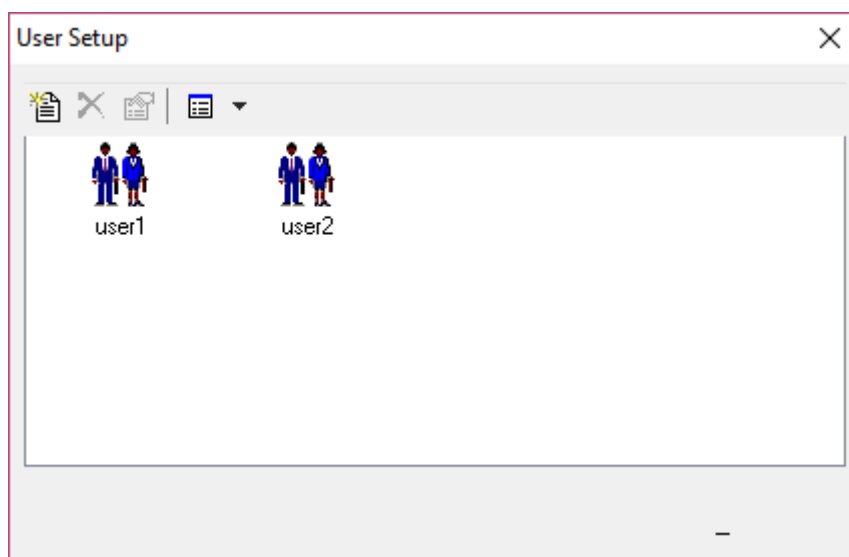
En primer lugar deberemos de crear un servidor de correo en nuestro ordenador con el software proporcionado por el profesor, como vemos en la siguiente imagen.



Introducimos el dominio.

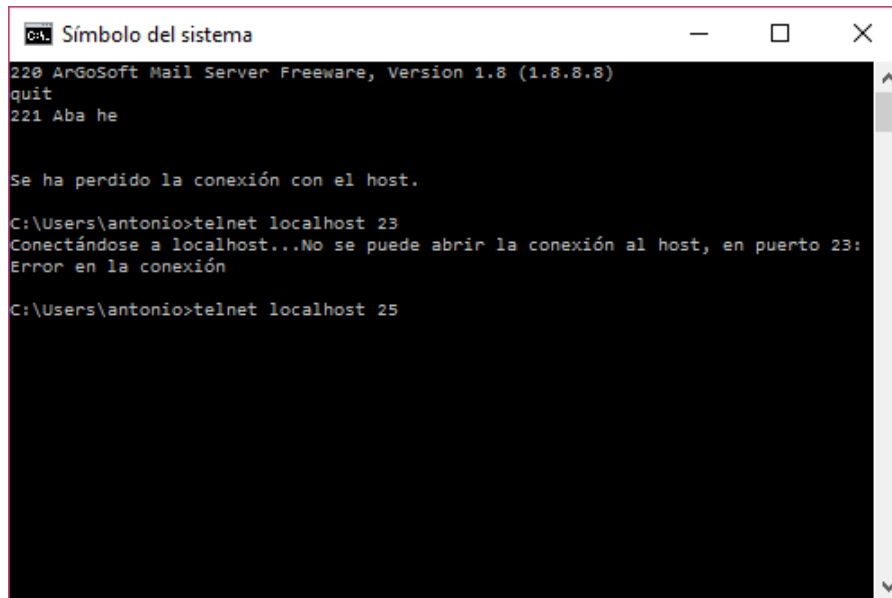


Introducimos los usuarios que podrán comunicarse en la red:



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.



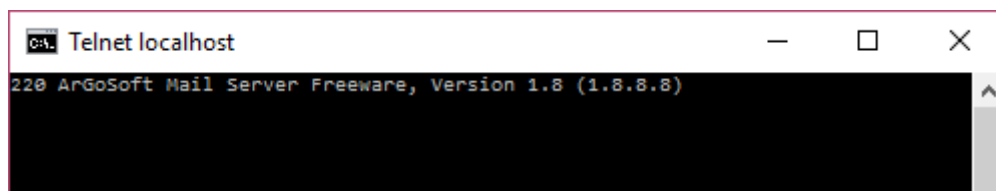
```
C:\> Símbolo del sistema
220 ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)
quit
221 Aba he

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
```

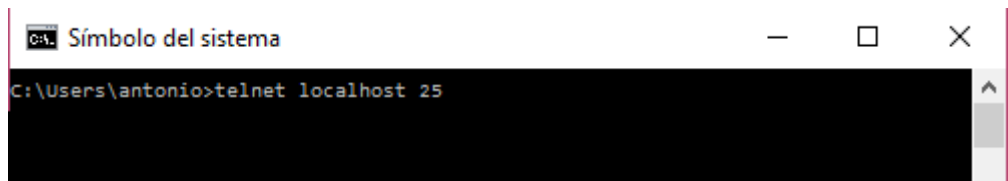
Introduciendo el puerto por defecto de SMTP, nos permite acceder al cliente Telnet.



```
C:\> 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

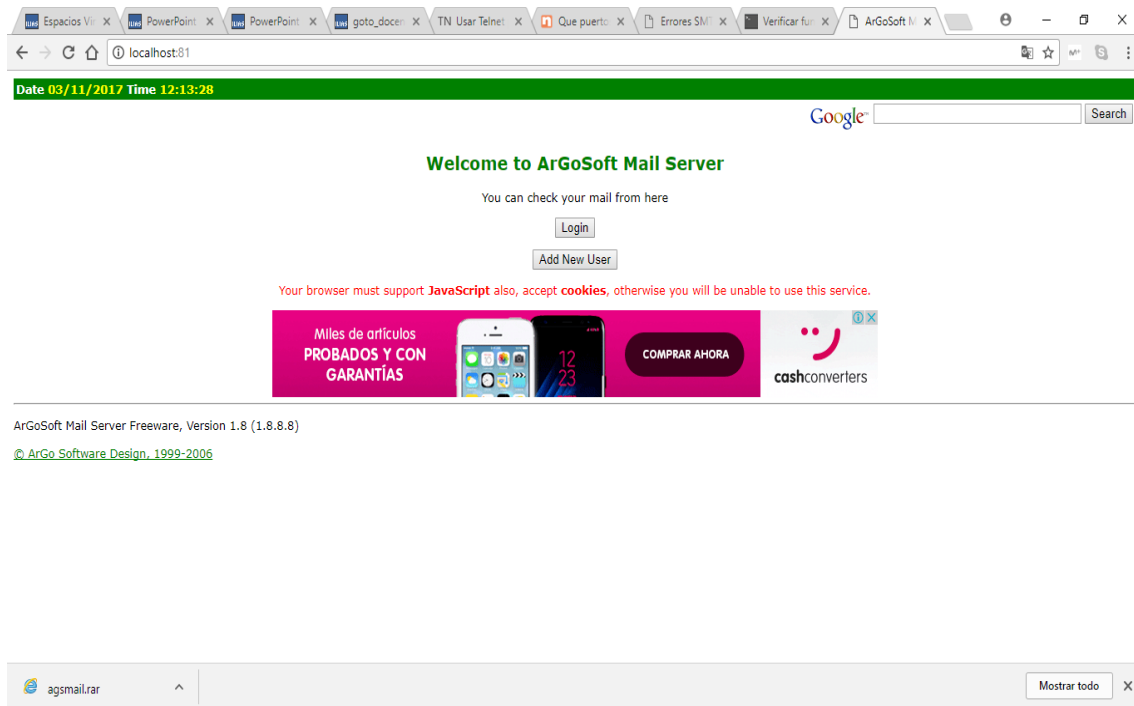
```
Telnet localhost
+OK ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)
USER user2
+OK Password required for user2
PASS Usuario2
+OK Mailbox locked and ready
LIST
+OK
1 239
2 265
.
RETR 2
+OK 265 octets
Received: from [127.0.0.1] by VAIO
(ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)); Fri, 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
.
```

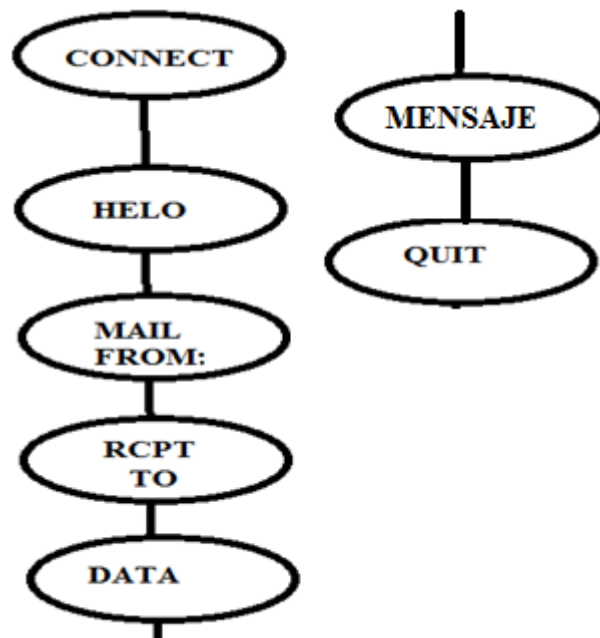
Nota:

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	
Logging	SMTP Authentication
General	Local Domains
Ports	
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
```

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

```
165         switch (estado) {
166             case S_HELO:
167                 // Se recibe el mensaje de bienvenida
168                 printf("\nBienvenido a SEVICEMAIL-ANFE\r\n");
169                 //sprintf_s da formato y almacena una serie de caracteres y valores en un buffer
170                 //Escribe en el servido HELO y el ipdest
171                 sprintf_s(buffer_out, sizeof(buffer_out), "%s %s %s", HE, ipdest, CRLF); //250 correcto
172                 estado++;
173                 break;
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

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
```

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:

```
176 case S_MAIL:
177     printf("MAIL FROM (enter para salir): ");
178     //Lee los caracteres de la entrada estándar y los almacena como una
179     //cadena hasta que se alcanza un carácter de nueva línea
180     gets_s(input, sizeof(input));
181     if (strlen(input) == 0) {
182         // Si la longitud de input es 0,
183         //Escribe en el servidor QUIT y pasamos al estado QUIT
184         sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", SD, CRLF); //SD -> QUIT
185         estado = S_QUIT;
186     }
187     else {
188         //Escribe en el servidor MAIL e input y pasamos al siguiente estado
189         sprintf_s(buffer_out, sizeof(buffer_out), "%s %s%s", MA, input, CRLF);
190         estado++;
191     }
192     break;
193
194
195 case S_RCPT:
196     printf("RCPT TO (enter para salir): ");
197     //Lee los caracteres de la entrada estándar y los almacena como una
198     //cadena hasta que se alcanza un carácter de nueva línea
199     gets_s(input2, sizeof(input2));
200
201     if (strlen(input2) == 0) {
202         // Si la longitud de input es 0,
203         //Escribe en el servidor QUIT y pasamos al estado QUIT
204         sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", SD, CRLF); //SD -> QUIT
205         estado = S_QUIT;
206     }
207     else {
208         //Escribe en el servidor RCPT TO: input y crlf y pasamos al siguiente estado
209         sprintf_s(buffer_out, sizeof(buffer_out), "%s%s%s", RCPT, input2, CRLF);
210
211         estado++;
212     }
213     break;
```

```

214
215         case S_DATA:
216             /* MAL
217             printf("CLIENTE> Introduzca datos (enter para salir): ");
218             //Introducimos datos
219             gets_s(input4, sizeof(input4));
220             //Si el tamaño de input4 es 0, nos salimos
221             if (strlen(input4) == 0) {
222                 sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", SD, CRLF);
223                 estado = S_QUIT;
224             }
225             else { /*
226
227                 //Escribe en el servidor DATA y pasamos al siguiente estado
228                 sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", DATA, CRLF);
229                 estado++;
230             break;

```

```

232
233         case S_MENSAJE:
234             printf("Escribe mensaje(enter para finalizar): ");
235             //Lee los caracteres de la entrada estándar y los almacena como una
236             //cadena hasta que se alcanza un carácter de nueva línea
237             gets(input5, sizeof(input5));
238             do {
239                 printf("Introduce un punto: ");
240                 gets(input4, sizeof(input4));
241             } while (strcmp(input4, ".") != 0);
242
243             //Escribimos asunto
244             printf("Asunto:");
245             gets(input3);
246             //Escribimos el mensaje
247             printf("\nMensaje de correo: \r\n");
248             printf("Fecha y hora: %s \r\n", salida_tiempo); //Nos da la hora en la que se ha enviado
249             printf("Asunto: %s%s", input3, CRLF);
250             printf("Remitente: %s%s%s", MA, input, CRLF);
251             printf("Destinatario: %s%s%s", RCPT, input2, CRLF);
252             printf("Datos: %s%s", input5, CRLF);
253             //Escribe en el servidor
254             sprintf_s(buffer_out, sizeof(buffer_out), "Date:%s%s From:%s%s To:%s%s Subject:%s%s DATA: %s%s%s%s", salida_tiempo, CRLF, input, CRLF,
255                 input2, CRLF, input3, CRLF, input5, CRLF, input4, CRLF);
256
257             estado++;
258             printf("SERVIDOR> Datos enviados correctamente\r\n");
259
260             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:

```
293
294
295     recibidos=recv(sockfd,buffer_in,512,0);
296     if(recibidos<=0){
297         DWORD error=GetLastError();
298         if(recibidos<0){
299             printf("CLIENTE> Error %d en la recepción de datos\r\n",error);
300             estado=S_QUIT;
301         }
302         else{
303             printf("CLIENTE> Conexión con el servidor cerrada\r\n");
304             estado=S_QUIT;
305         }
306     }
307     else {
308         buffer_in[recibidos] = 0x00;
309         //Escribe un mensaje de envío correcto y recepción correcta
310         printf(buffer_in);
311         //Definimos en protocol.h una respuesta a un comando de aplicacion
312         //Será UU y significa: "554 User unknown"
313         //Si el buffer_in es usuario erroneo
314         if (strncmp(buffer_in, UU, 2) == 0) {
315             //Definimos variable para introducir enteros
316             int estado2=0;
317             do {
318                 printf("Introduce 1 --> Introducir los dos usuarios de nuevo\n2--> Para introducir un usuario correcto\n");
319                 //scanf_s necesita el tipo de dato %i y direccion &.
320                 scanf_s("%i", &estado2);
321                 switch (estado2) {
322                     case 1:
323                         estado = S_MAIL;
324                         //Nos permite poder escribir de nuevo en el case S_MAIL
325                         gets_s(input, sizeof(input));
326                         break;
327
328                     case 2:
329                         estado = S_RCPT;
330
331                         gets_s(input2, sizeof(input2));
332                         break;
333
334                     default:
335                         printf("Opcion no disponible\n");
336                         break;
337                 }
338             } while (estado2 != 1 && estado2 != 2);
339         }
340     }
341 }
342 }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
44
```

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:

```
260         case S_RSET:
261             do {
262                 printf("¿Desea escribir otro mensaje? (s/n)\r\n");
263                 caracter = _getche(); //Lee caracter
264             } while (caracter != 's' && caracter != 'n' && caracter != 'S' && caracter != 'N');
265             //Si es S o s pasamos al estado S_HELO y escribimos otro mensaje
266             if (caracter == 'S' || caracter == 's') {
267                 //Escribe en el servidor RSET y pasamos al estado HELO
268                 sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", RSET, CRLF);
269                 //Enviamos el mensaje mediante el comando send de TCP
270                 enviados = send(sockfd, buffer_out, (int)strlen(buffer_out), 0);
271                 estado = S_HELO;
272             }
273             else {
274                 //Si no, pasamos al estado S_QUIT y nos salimos
275                 sprintf_s(buffer_out, sizeof(buffer_out), "%s%s", SD, CRLF);
276                 estado++;
277             }
278             break;
```

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 (SESSION 5)
37     struct hostent *host;
38     struct in_addr address; //SESSION 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;
```

Quinta sesión:

Pruebas.

Añadir la resolución de dominios.

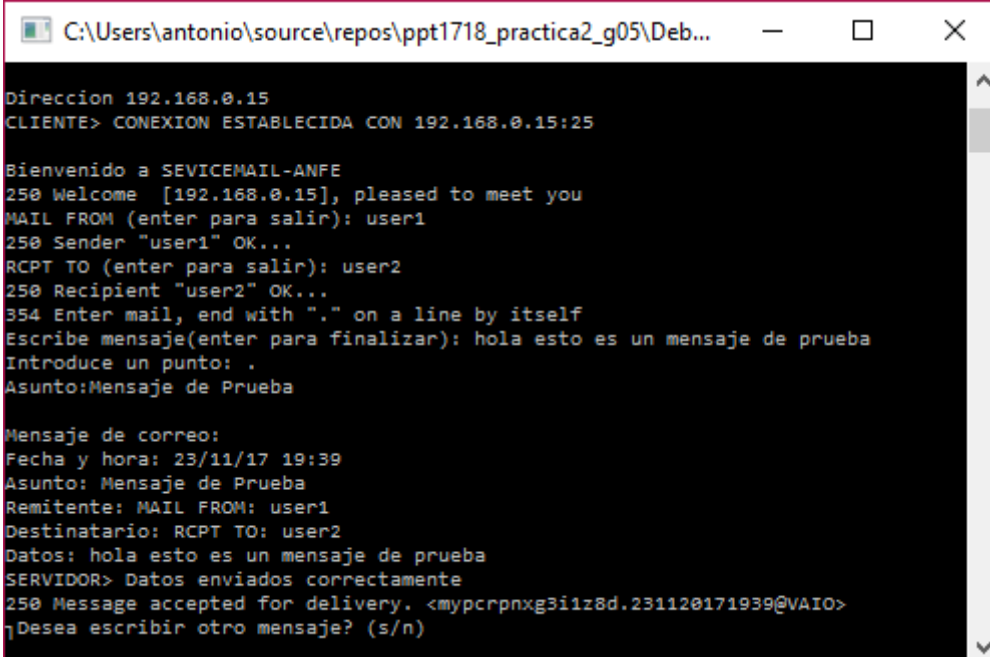
Para añadir la resolución a dominios hemos utilizado el código proporcionado por el profesor, haciendo un **strcpy** adicional, simplemente para copiar en **ipdest** la dirección de **address**

```
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     }
```

Las pruebas pueden ser comprobadas mediante el cliente TELNET.

Veamos un ejemplo de envío de mensaje y de comprobación:

Enviamos un mensaje de prueba




```
C:\Users\antonio\source\repos\ppt1718_practica2_g05\Deb...
Dirección 192.168.0.15
CLIENTE> CONEXION ESTABLECIDA CON 192.168.0.15:25

Bienvenido a SEVICEMAIL-ANFE
250 Welcome [192.168.0.15], pleased to meet you
MAIL FROM (enter para salir): user1
250 Sender "user1" OK...
RCPT TO (enter para salir): user2
250 Recipient "user2" OK...
354 Enter mail, end with "." on a line by itself
Escribe mensaje(enter para finalizar): hola esto es un mensaje de prueba
Introduce un punto: .
Asunto:Mensaje de Prueba


Mensaje de correo:
Fecha y hora: 23/11/17 19:39
Asunto: Mensaje de Prueba
Remitente: MAIL FROM: user1
Destinatario: RCPT TO: user2
Datos: hola esto es un mensaje de prueba
SERVIDOR> Datos enviados correctamente
250 Message accepted for delivery. <mypcrpnxg3i1z8d.231120171939@VAIO>
¿Desea escribir otro mensaje? (s/n)
```


Y accedemos al cliente telnet para comprobar si el envío del mensaje se ha realizado correctamente:

 Símbolo del sistema

```
Microsoft Windows [Versión 10.0.14393]
(c) 2016 Microsoft Corporation. Todos los derechos reservados.


C:\Users\antonio>telnet
```

 Símbolo del sistema - telnet

```
Cliente Telnet de Microsoft

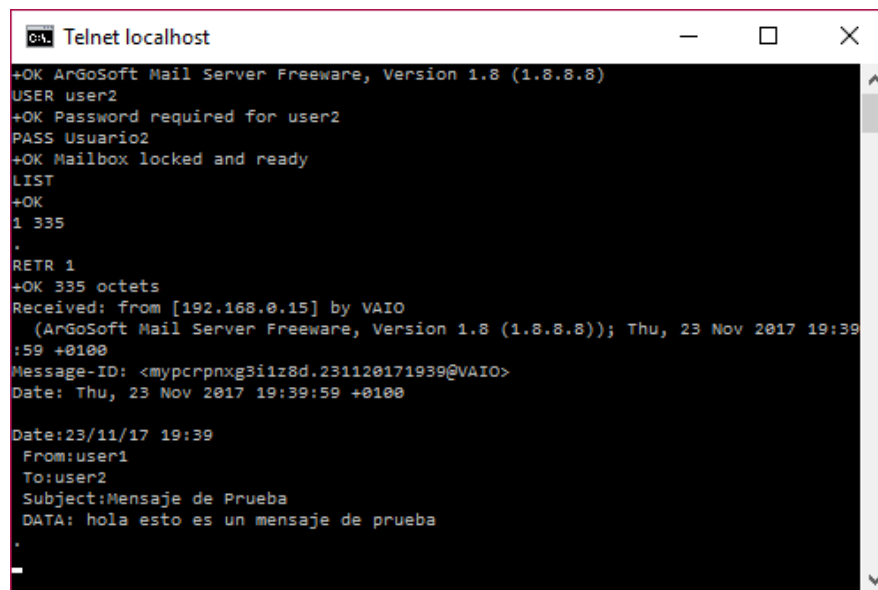
El carácter de escape es "CTRL++"

Microsoft Telnet> o localhost 110
```

 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
1 335
.
```

Como podemos observar tenemos un mensaje, comprobamos si es el que hemos enviado anteriormente.



```
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
1 335
.
RETR 1
+OK 335 octets
Received: from [192.168.0.15] by VAI0
      (ArGoSoft Mail Server Freeware, Version 1.8 (1.8.8.8)); Thu, 23 Nov 2017 19:39:59 +0100
Message-ID: <mypcrpnxg3i1z8d.231120171939@VAIO>
Date: Thu, 23 Nov 2017 19:39:59 +0100

Date:23/11/17 19:39
From:user1
To:user2
Subject:Mensaje de Prueba
DATA: hola esto es un mensaje de prueba
.
```

Protocolos de transporte

Antonio Osuna Melgarejo

Fernando Cabrera Caballero

3ºCurso

Ingeniería Telemática

