

Seminario de Redes de Computadoras 66.48

DESARROLLO DE APLICATIVOS CON WINSOCKETS

*Docentes: Ing. Marcelo Utard
Ing. Pablo Ronco*

*Alumnos: Baños, Germán
Gámez, Pablo
Rabino, Juan Pablo
Salas, Federico*

Introducción

En el siguiente trabajo nos proponemos lograr una introducción completa a los sockets en entorno Windows para lograr luego generar aplicaciones prácticas de lo aprendido a lo largo del curso.

¿Qué es un socket?

Los sockets son una interfaz de entrada-salida que permite la comunicación entre procesos en el mismo equipo o en equipos diferentes.

Los sockets brindan al programa de aplicación la capacidad de acceder a las funciones del sistema operativo, mediante "system calls". Básicamente en el sentido práctico es una forma de independizar al programador de las capas inferiores de comunicación. Simplemente los sockets se encargan de la comunicación física, empaquetamiento de los datagramas, handshaking tcp, etc.

Cada socket que se crea queda totalmente especificado mediante una dirección IP, un protocolo de comunicación (i.e. TCP, UDP) y un número de puerto.

Winsock

En la fig 1 se ve claramente el esquema de funcionamiento de la interfaz de sockets.

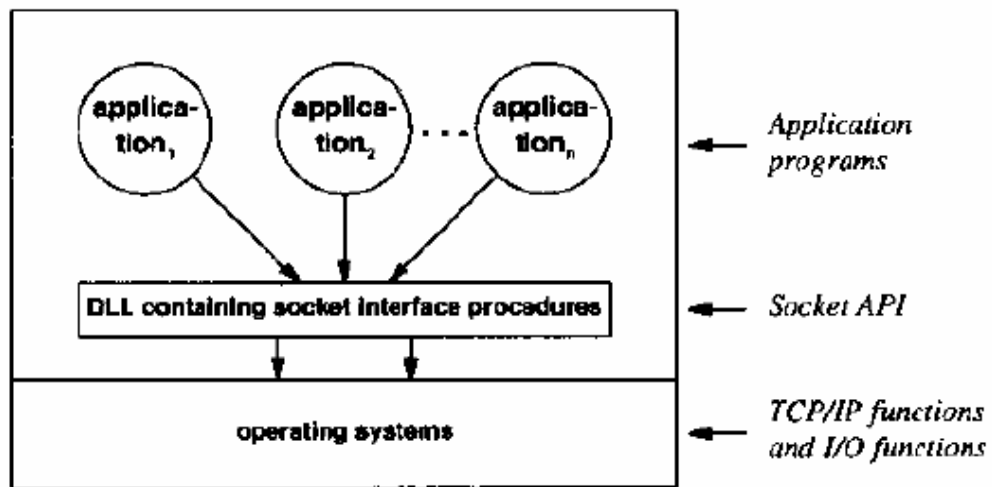


Fig.1 Esquema de interfaces en sockets

En un sentido más práctico cuando se realiza el programa no se incluyen todas las funciones de la API Winsock sino que se incluyen en una biblioteca (DLL) que se carga dinámicamente en tiempo de ejecución. Y varias aplicaciones pueden compartir dichas funciones de biblioteca para no ser cargadas nuevamente con cada programa.

Funciones de la Winsock API

WSAStartup: Inicializa la librería de Windows Sockets. Se produce un enlace a la misma en tiempo de ejecución.

WSACleanup: Se debe ejecutar esta función al terminar el programa para quitar el enlace del proceso aplicativo a la DLL.

Socket: Crea un nuevo socket. Se le pasa el nombre de la familia de protocolos (ej.:TCP/IP) y el tipo de servicio (TCP o UDP). Devuelve un descriptor de archivo.

int socket (int dominio, int tipo, int protocolo);

Connect: Establece una conexión con un servidor remoto. Se le pasa el IP y el puerto de la máquina remota.

int connect (int sock, struct sockaddr *servidor, size_t servidor);

Bind: Asocia al socket una dirección IP y un número de puerto.

int bind (int sock, struct sockaddr *servidor, size_t servidor);

Send: Envía información. Se tiene que especificar descriptor de archivo, dirección de memoria de los datos a enviar, tamaño del buffer y bits para control de transmisión.

int send (int sock, const void *mensaje, size_t mensaje, int flags);

Recv: Recibe información. Es análoga a Send.

int recv (int sock, void *mensaje, size_t mensaje, int flags);

Listen: Acepta nuevas conexiones entrantes al servidor. Necesita como parámetros el tipo de socket (activo=cliente o pasivo=servidor) y tamaño de cola de direcciones entrantes.

int listen (int sock, n_conexiones);

Accept: Acepta una conexión. Se le pasa el descriptor del socket entrante y crea un nuevo socket para cada conexión.

int accept (int sock, struct sockaddr *servidor, inet *tam_sockaddr);

Primera implementación: Cliente HTTP

Se desarrolló una pequeña aplicación capaz de obtener un objeto "index.html" a partir de una dirección IP mediante la conexión con el puerto 80 del Server y el comando GET de HTTP.

IP: 72.14.205.99 (www.google.com)

Puerto: 80

Comando: GET /index.html HTTP/1.1\nAccept: */*\n\n

Obteniendo lo siguiente:

```

ok
Solicitando informaci3n
Presione cualquier tecla para continuar . . .

HTTP/1.1 302 Found
Location: http://www.google.com.ar/index.html
Cache-Control: private
Set-Cookie: PREF=ID=7027229cac42c5de:TM=1196088547:LM=1196088547:S=z1xK1hp31qMer
uLc; expires=Wed, 25-Nov-2009 14:49:07 GMT; path=/; domain=.google.com
Content-Type: text/html
Server: gws
Content-Length: 232
Date: Mon, 26 Nov 2007 14:49:07 GMT

<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.com.ar/index.html">here</A>.
</BODY></HTML>
Presione cualquier tecla para continuar . . .

```

Podemos observar que la pagina nos redirecciona a www.google.com.ar/index.html. Pero de todas maneras la aplicaci3n logr3 su prop3sito.

En la siguiente captura podemos ver el datagrama que solicita el objeto `index.html`

```

+ Frame 11 (92 bytes on wire, 92 bytes captured)
+ Ethernet II, Src: Pro-Nets_0e:c6:f6 (00:06:4f:0e:c6:f6), Dst: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e)
+ Internet Protocol, Src: 192.168.0.2 (192.168.0.2), Dst: 72.14.205.99 (72.14.205.99)
  Version: 4
  Header length: 20 bytes
  + Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 78
  Identification: 0x8488 (33928)
  + Flags: 0x04 (Don't Fragment)
  Fragment offset: 0
  Time to live: 128
  Protocol: TCP (0x06)
  + Header checksum: 0xa005 [correct]
  Source: 192.168.0.2 (192.168.0.2)
  Destination: 72.14.205.99 (72.14.205.99)
+ Transmission Control Protocol, Src Port: 120 (120), Dst Port: http (80), Seq: 1, Ack: 1, Len: 38
  Source port: 120 (120)
  Destination port: http (80)
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 39 (relative sequence number)]
  Acknowledgement number: 1 (relative ack number)
  Header length: 20 bytes
  + Flags: 0x18 (PSH, ACK)
  Window size: 17160
  + Checksum: 0x17f9 [correct]
+ Hypertext Transfer Protocol
+ GET /index.html HTTP/1.1\n
  Request Method: GET
  Request URI: /index.html
  Request Version: HTTP/1.1
  Accept: */*\n
  \n

```

Y luego el datagrama enviado por el servidor redireccionando a www.google.com.ar

```

⊞ Frame 15 (627 bytes on wire, 627 bytes captured)
⊞ Ethernet II, Src: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e), Dst: Pro-Nets_0e:c6:f6 (00:06:4f:0e:c6:f6)
⊞ Internet Protocol, Src: 72.14.205.99 (72.14.205.99), Dst: 192.168.0.2 (192.168.0.2)
    Version: 4
    Header length: 20 bytes
    ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        Total Length: 613
        Identification: 0xd2c7 (53959)
    ⊞ Flags: 0x00
        Fragment offset: 0
        Time to live: 50
        Protocol: TCP (0x06)
    ⊞ Header checksum: 0xddaf [correct]
        Source: 72.14.205.99 (72.14.205.99)
        Destination: 192.168.0.2 (192.168.0.2)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 120 (120), Seq: 1, Ack: 39, Len: 573
    Source port: http (80)
    Destination port: 120 (120)
    Sequence number: 1 (relative sequence number)
    [Next sequence number: 574 (relative sequence number)]
    Acknowledgement number: 39 (relative ack number)
    Header length: 20 bytes
    ⊞ Flags: 0x18 (PSH, ACK)
    Window size: 5720
    ⊞ Checksum: 0x7254 [correct]
⊞ Hypertext Transfer Protocol
    ⊞ HTTP/1.1 302 Found\r\n
        Request Version: HTTP/1.1
        Response Code: 302
        Location: http://www.google.com.ar/index.html\r\n
        Cache-Control: private\r\n
        Set-Cookie: PREF=ID=2eeb1d5f18b2b43b:TM=1196087705:LM=1196087705:S=FKDwzWIoHCo9cdz6; expires=wed, 25-Nov-2009 14:35:05 GMT;
        Content-Type: text/html\r\n
        Server: gws\r\n
        Content-Length: 232
        Date: Mon, 26 Nov 2007 14:35:05 GMT\r\n
        \r\n
    ⊞ Line-based text data: text/html
        <HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">\n
        <TITLE>302 Moved</TITLE></HEAD><BODY>\n
        <H1>302 Moved</H1>\n
        The document has moved\n
        <A HREF="http://www.google.com.ar/index.html">here</A>.\r\n
        </BODY></HTML>\r\n

```

Segunda implementación: DNS

Este programa tomará como argumento por la línea de comandos el nombre de un equipo y nos dirá su nombre oficial y su dirección ip principal, de no encontrarlo nos dará un mensaje de error.

Código fuente:

```

#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>
#include <windows.h>

int main(int argc, char *argv[])
{
    WSADATA datos; // datos que usa el winsock
    WORD winsock; // versión de winsock a utilizar
    struct hostent *info; // información que obtendremos del equipo

    // Comprobamos que se metan los parametros adecuados
    if(argc!=2)
        printf("Uso: resuelveNombre <nombre_equipo> \n");
    else
    {
        // Usaremos la versión 1.0 de winsock
        winsock=MAKEWORD(1,1);
        // Inicializamos la libreria winsock llamando a WSStartup
        if((WSStartup(winsock, &datos))===-1)
            printf("Error: Problemas al inicializar el winsock \n");
        else
        {
            // Obtenemos información del equipo
            info=gethostbyname(argv[1]);
            /* Si el equipo no existe o no se puede resolver la dirección
            lo indicamos, sino visualizamos los datos pedidos: ip + nombre oficial*/
            if(info==NULL) // es lo mismo que if(!info)
                printf("Error: No se puede resolver \"%s\" \n", argv[1]);
            else
            {
                // Visualizamos los datos almacenados en la estructura info
                printf("Nombre Oficial: %s \n", info->h_name);
                /* La ip está en formato de red, hay que pasarla a formato ASCII con
                la función inet_ntoa */
                printf("Ip Principal: %s \n", inet_ntoa(*(struct in_addr *)info->h_addr));
            }
            // Cerramos la libreria winsock
            WSACleanup();
        }
    }

    system("PAUSE");
    EXIT_SUCCESS;
}

```

Con lo que obtuvimos el siguiente resultado al ingresar como argumento www.ole.com.ar

```

C:\Users\PC\Documents\2° Cuatrimestre 2007\Seminario Redes\tp4\Proyecto 1.exe
Nombre Oficial: www.ole.clarin.com
Ip Principal: 200.42.136.189
Presione una tecla para continuar . . . _

```

Y en la captura pudimos evidenciar el uso del query DNS

No. -	Time	Source	Destination	Protocol	Info
14	54.474610	201.252.58.32	200.45.191.35	DNS	Standard query A www.ole.com.ar
15	54.489258	200.45.191.35	201.252.58.32	DNS	Standard query response CNAME www.ole.clarin.com A 200.42.136.189

```

# Frame 14 (74 bytes on wire, 74 bytes captured)
# Ethernet II, Src: Xerox_00:00:00 (00:00:01:00:00:00), Dst: 54:72:20:00:01:00 (54:72:20:00:01:00)
# Internet Protocol, Src: 201.252.58.32 (201.252.58.32), Dst: 200.45.191.35 (200.45.191.35)
# User Datagram Protocol, Src Port: 3185 (3185), Dst Port: domain (53)
  Source port: 3185 (3185)
  Destination port: domain (53)
  Length: 40
  Checksum: 0x9439 [correct]
# Domain Name System (query)
  Transaction ID: 0x0059
  # Flags: 0x0100 (Standard query)
    0... .. = Response: Message is a query
    .000 0... .. = Opcode: Standard query (0)
    .... ..0. .... = Truncated: Message is not truncated
    .... ..1 .... = Recursion desired: Do query recursively
    .... ..0. .... = Z: reserved (0)
    .... ..0 .... = Non-authenticated data OK: Non-authenticated data is unacceptable
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  # Queries
    # www.ole.com.ar: type A, class IN
      Name: www.ole.com.ar
      Type: A (Host address)
      Class: IN (0x0001)

```

0000	54	72	20	00	01	00	00	00	01	00	00	00	08	00	45	00	TrE.
0010	00	3c	42	42	00	00	80	11	6d	01	c9	fc	3a	20	c8	2d	.	<BB...	m...: :-
0020	bf	23	0c	71	00	35	00	28	94	39	00	59	01	00	00	01	.	#.q.5.(.9.Y...
0030	00	00	00	00	00	03	77	77	77	03	6f	6c	65	03	63	w	ww.ole.c	
0040	6f	6d	02	61	72	00	00	01	00	01							om.ar...	..	

Y la correspondiente respuesta del servidor

No. -	Time	Source	Destination	Protocol	Info
14	54.474610	201.252.58.32	200.45.191.35	DNS	Standard query A www.ole.com.ar
15	54.489258	200.45.191.35	201.252.58.32	DNS	Standard query response CNAME www.ole.clarin.com A 200.42.136.189

```

# Frame 15 (122 bytes on wire, 122 bytes captured)
# Ethernet II, Src: 54:72:20:00:01:00 (54:72:20:00:01:00), Dst: Xerox_00:00:00 (00:00:01:00:00:00)
# Internet Protocol, Src: 200.45.191.35 (200.45.191.35), Dst: 201.252.58.32 (201.252.58.32)
# User Datagram Protocol, Src Port: domain (53), Dst Port: 3185 (3185)
# Domain Name System (response)
  Transaction ID: 0x0059
  # Flags: 0x8180 (Standard query response, No error)
  Questions: 1
  Answer RRs: 2
  Authority RRs: 0
  Additional RRs: 0
  # Queries
    # www.ole.com.ar: type A, class IN
      Name: www.ole.com.ar
      Type: A (Host address)
      Class: IN (0x0001)
  # Answers
    # www.ole.com.ar: type CNAME, class IN, cname www.ole.clarin.com
      Name: www.ole.com.ar
      Type: CNAME (Canonical name for an alias)
      Class: IN (0x0001)
      Time to live: 8 hours, 12 minutes, 40 seconds
      Data length: 20
      Primary name: www.ole.clarin.com
    # www.ole.clarin.com: type A, class IN, addr 200.42.136.189
      Name: www.ole.clarin.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 12 minutes, 47 seconds
      Data length: 4
      Addr: 200.42.136.189

```

0020	3a	20	00	35	0c	71	00	58	bd	39	00	59	81	80	00	01	:	5.q.X	.9.Y...
0030	00	02	00	00	00	00	03	77	77	77	03	6f	6c	65	03	63	w	ww.ole.c
0040	6f	6d	02	61	72	00	00	01	00	01	c0	0c	00	05	00	01		om.ar...
0050	00	00	73	78	00	14	03	77	77	77	03	6f	6c	65	06	63		..sx...	ww.ole.c
0060	6c	61	72	69	6e	03	63	6f	6d	00	c0	2c	00	01	00	01		larin.co	m.....
0070	00	00	02	ff	00	04	c8	2a	88	bd									

Luego decidimos probar con un caso incorrecto al pasarle el argumento www.oole.coml.ar

C:\Users\PC\Documents\2° Cuatrimestre 2007\Seminaro Redes\tp4\Proyecto 1.exe

Error: No se puede resolver "www.oole.com.ar"
Presione una tecla para continuar . . .

Y las capturas

No. -	Time	Source	Destination	Protocol	Info
22	108.695313	201.252.58.32	200.45.191.35	DNS	Standard query A www.oole.com.ar
23	108.710938	200.45.191.35	201.252.58.32	DNS	Standard query response, No such name
+ Frame 22 (75 bytes on wire, 75 bytes captured)					
+ Ethernet II, Src: Xerox_00:00:00 (00:00:01:00:00:00), Dst: 54:72:20:00:01:00 (54:72:20:00:01:00)					
+ Internet Protocol, Src: 201.252.58.32 (201.252.58.32), Dst: 200.45.191.35 (200.45.191.35)					
- User Datagram Protocol, Src Port: 3185 (3185), Dst Port: domain (53)					
Source port: 3185 (3185)					
Destination port: domain (53)					
Length: 41					
Checksum: 0xde7b [correct]					
- Domain Name System (query)					
Transaction ID: 0x005a					
- Flags: 0x0100 (Standard query)					
0... .. = Response: Message is a query					
.000 0... .. = Opcode: Standard query (0)					
.... ..0. = Truncated: Message is not truncated					
.... ..1 = Recursion desired: Do query recursively					
.... ..0. = Z: reserved (0)					
.... ..0 = Non-authenticated data OK: Non-authenticated data is unacceptable					
Questions: 1					
Answer RRs: 0					
Authority RRs: 0					
Additional RRs: 0					
- Queries					
- www.oole.com.ar: type A, class IN					
Name: www.oole.com.ar					
Type: A (Host address)					
Class: IN (0x0001)					
0010	00 3d 42 43 00 00 80 11	6c ff c9 fc 3a 20 c8 2d	..=BC.... l...: :-		
0020	bf 23 0c 71 00 35 00 29	de 7b 00 5a 01 00 00 01	..#.q.5.)..Z....		
0030	00 00 00 00 00 00 03 77	77 77 04 6f 6f 6c 65 03w ww.oole.		
0040	63 6f 6d 02 61 72 00 00	01 00 01	com.ar.. ...		

No.	Time	Source	Destination	Protocol	Info
22	108.695313	201.252.58.32	200.45.191.35	DNS	Standard query A www.oole.com.ar
23	108.710938	200.45.191.35	201.252.58.32	DNS	Standard query response, No such name


```

⊕ Frame 23 (130 bytes on wire, 130 bytes captured)
⊕ Ethernet II, Src: 54:72:20:00:01:00 (54:72:20:00:01:00), Dst: Xerox_00:00:00 (00:00:01:00:00:00)
⊕ Internet Protocol, Src: 200.45.191.35 (200.45.191.35), Dst: 201.252.58.32 (201.252.58.32)
⊕ User Datagram Protocol, Src Port: domain (53), Dst Port: 3185 (3185)
- Domain Name System (response)
  Transaction ID: 0x005a
  ⊕ Flags: 0x8183 (Standard query response, No such name)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 1
    Additional RRs: 0
  ⊖ Queries
    ⊖ www.oole.com.ar: type A, class IN
      Name: www.oole.com.ar
      Type: A (Host address)
      Class: IN (0x0001)
    ⊖ Authoritative nameservers
      ⊖ com.ar: type SOA, class IN, mname athea.ar
        Name: com.ar
        Type: SOA (Start of zone of authority)
        Class: IN (0x0001)
        Time to live: 3 hours
        Data length: 43
        Primary name server: athea.ar
        Responsible authority's mailbox: noc-ar.atina.ar
        Serial number: 2007112201
        Refresh interval: 6 hours
        Retry interval: 1 hour
        Expiration limit: 20 days
        Minimum TTL: 6 hours
  
```



```

0020 3a 20 00 35 0c 71 00 60 1f 0e 00 5a 81 83 00 01  : .5.q. .z...
0030 00 00 00 01 00 00 03 77 77 77 04 6f 6f 6c 65 03  : .....w ww.oole.
0040 63 6f 6d 02 61 72 00 00 01 00 01 c0 15 00 06 00  : com.ar. ....
0050 01 00 00 2a 30 00 2b 05 61 74 68 65 61 c0 19 06  : ...*0+. athea...
0060 6e 6f 63 2d 61 72 05 61 74 69 6e 61 c0 19 77 a2  : noc-ar.a tina.w.
0070 15 00 00 00 54 72 00 00 00 00 00 00 00 00 00 00  : .....
  
```

Tercera implementación: DNS inverso

Ahora lo que haremos es la implementación inversa del problema anterior, pasaremos una dirección IP y el programa nos devolverá el nombre del dominio.

Código Fuente:

```

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock.h>

int main(int argc, char *argv[])
{
    WSADATA datos; // datos que usa el winsock
    WORD winsock; // versión de winsock a utilizar
    struct hostent *info; // información que obtendremos del equipo
    long ip; // La ip pasada la almacenaremos en formato long

    // Comprobamos que se metan los parametros adecuados
    if(argc!=2)
        printf("Uso: resuelveIp <ip_equipo> \n");
    else
    {
        // Usaremos la versión 1.0 de winsock
        winsock=MAKEWORD(1,1);
        // Inicializamos la libreria winsock llamando a WSStartup
        if((WSStartup(winsock, &datos))!=-1)
            printf("Error: Problemas al inicializar el winsock \n");
        else
  
```

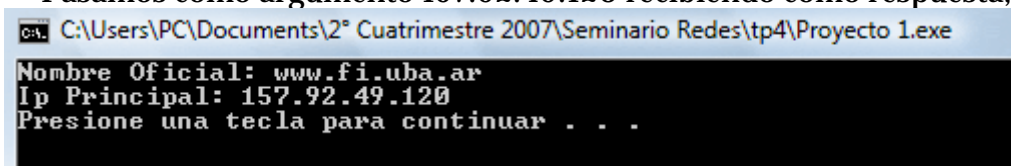
```

{
    /* Pasamos la cadena de caracteres de la ip a formato long. Para
       ello deberemos utilizar la funcion inet_addr */
    ip=inet_addr(argv[1]);
    /* Obtenemos información del equipo con la funcion gethostbyaddr */
    info=gethostbyaddr((char *)&ip, sizeof(ip), AF_INET);

    /* Si el equipo no existe o no se puede resolver la dirección
       lo indicamos, sino visualizamos los datos pedidos: ip + nombre oficial*/
    if(!info)
        printf("Error: No se puede resolver \"%s\" \n", argv[1]);
    else
    {
        // Visualizamos los datos almacenados en la estructura info
        printf("Nombre Oficial: %s \n", info->h_name);
        /* La ip está en formato de red, hay que pasarla a formato ASCII con
           la función inet_ntoa */
        printf("Ip Principal: %s \n", inet_ntoa(*(struct in_addr *)info->h_addr));
    }
    // Cerramos la liberia winsock
    WSACleanup();
}
}

```

Pasamos como argumento 157.92.49.120 recibiendo como respuesta,



Podemos observar en la interfase el query dns

No. .	Time	Source	Destination	Protocol	Info
76	118.019362	192.168.0.2	200.45.191.35	DNS	Standard query PTR 120.49.92.157.in-addr.arpa
77	118.034556	200.45.191.35	192.168.0.2	DNS	Standard query response PTR www.fi.uba.ar

# Frame 76 (86 bytes on wire, 86 bytes captured)					
# Ethernet II, Src: Pro-Nets_0e:c6:f6 (00:06:4f:0e:c6:f6), Dst: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e)					
# Internet Protocol, Src: 192.168.0.2 (192.168.0.2), Dst: 200.45.191.35 (200.45.191.35)					
# User Datagram Protocol, Src Port: 1971 (1971), Dst Port: domain (53)					
Source port: 1971 (1971)					
Destination port: domain (53)					
Length: 52					
Checksum: 0xb922 [correct]					
# Domain Name System (query)					
Transaction ID: 0x0001					
# Flags: 0x0100 (standard query)					
0... .. = Response: Message is a query					
.000 0... .. = Opcode: Standard query (0)					
.... .0. = Truncated: Message is not truncated					
.... ..1 = Recursion desired: Do query recursively					
....0.. = Z: reserved (0)					
....0 = Non-authenticated data OK: Non-authenticated data is unacceptable					
Questions: 1					
Answer RRs: 0					
Authority RRs: 0					
Additional RRs: 0					
# Queries					
# 120.49.92.157.in-addr.arpa: type PTR, class IN					
Name: 120.49.92.157.in-addr.arpa					
Type: PTR (Domain name pointer)					
Class: IN (0x0001)					

0020	bf 23 07 b3 00 35 00 34 b9 22	00 01 01 00 00 014.....
0030	00 00 00 00 00 00 03 31 32 30	02 34 39 02 39 321 20.49.92
0040	03 31 35 37 07 69 6e 2d 61 64	64 72 04 61 72 70	.157.in- addr.arp
0050	61 00 00 0c 00 01		a.....

No. -	Time	Source	Destination	Protocol	Info
78	130.944219	192.168.0.2	200.45.191.35	DNS	Standard query PTR 1.0.92.157.in-addr.arpa
79	130.962266	200.45.191.35	192.168.0.2	DNS	Standard query response, No such name
<pre> ⊟ Frame 78 (83 bytes on wire, 83 bytes captured) ⊟ Ethernet II, Src: Pro-Nets_0e:c6:f6 (00:06:4f:0e:c6:f6), Dst: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e) ⊟ Internet Protocol, Src: 192.168.0.2 (192.168.0.2), Dst: 200.45.191.35 (200.45.191.35) ⊟ User Datagram Protocol, Src Port: 1972 (1972), Dst Port: domain (53) Source port: 1972 (1972) Destination port: domain (53) Length: 49 Checksum: 0xc3bf [correct] ⊟ Domain Name System (query) Transaction ID: 0x0001 Flags: 0x0100 (Standard query) 0... .. = Response: Message is a query .000 0... .. = Opcode: Standard query (0) 0. = Truncated: Message is not truncated 1. = Recursion desired: Do query recursively 0. = Z: reserved (0) 0. = Non-authenticated data OK: Non-authenticated data is unacceptable Questions: 1 Answer RRs: 0 Authority RRs: 0 Additional RRs: 0 Queries ⊟ 1.0.92.157.in-addr.arpa: type PTR, class IN Name: 1.0.92.157.in-addr.arpa Type: PTR (Domain name pointer) Class: IN (0x0001) </pre>					
<pre> 0020 00 02 00 35 07 b4 00 68 4c c1 00 01 81 83 00 01 ...5...h L... 0030 00 00 00 01 00 00 01 31 01 30 02 39 32 03 31 351 .0.92.15 0040 37 07 69 6e 2d 61 64 64 72 04 61 72 70 61 00 00 7.in-add r.arpa.. 0050 0c 00 01 </pre>					

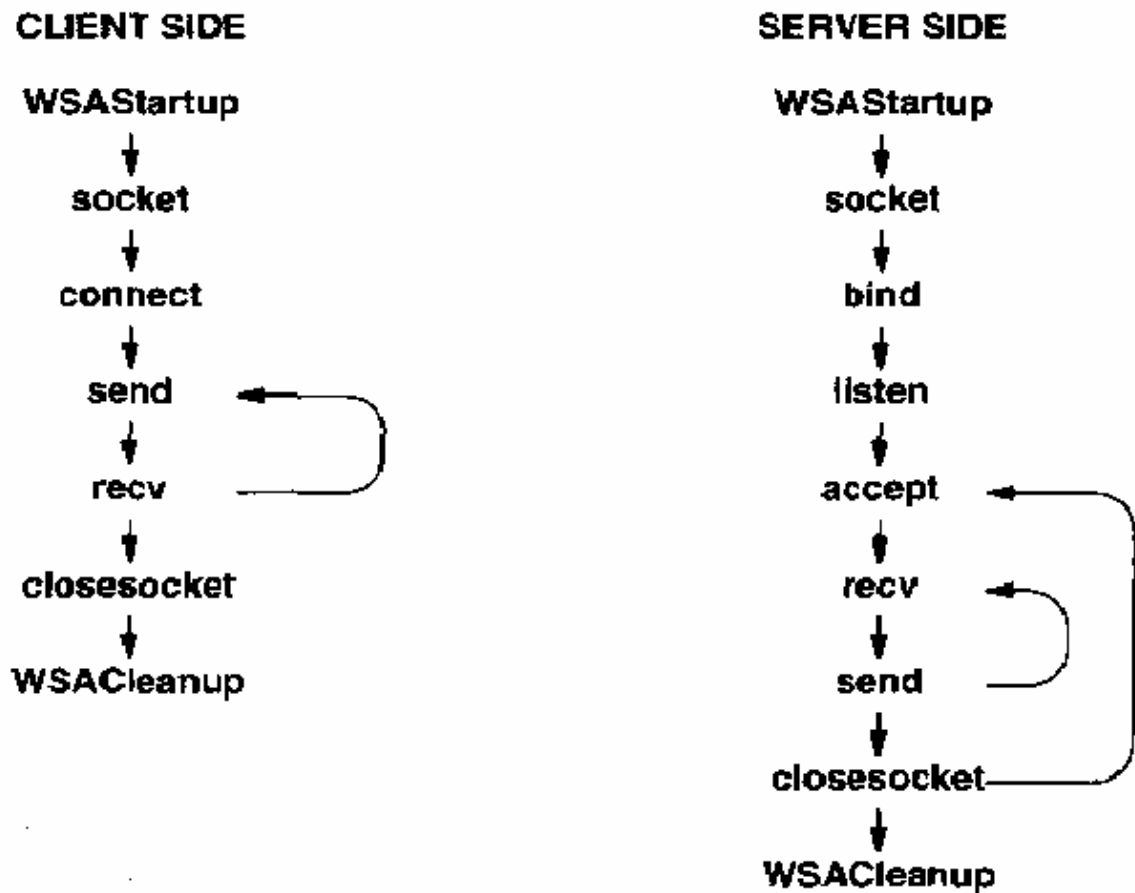
Y la respuesta del lado del servidor.

No. -	Time	Source	Destination	Protocol	Info
78	130.944219	192.168.0.2	200.45.191.35	DNS	Standard query PTR 1.0.92.157.in-addr.arpa
79	130.962266	200.45.191.35	192.168.0.2	DNS	Standard query response, No such name
<pre> ⊟ Frame 79 (138 bytes on wire, 138 bytes captured) ⊟ Ethernet II, Src: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e), Dst: Pro-Nets_0e:c6:f6 (00:06:4f:0e:c6:f6) ⊟ Internet Protocol, Src: 200.45.191.35 (200.45.191.35), Dst: 192.168.0.2 (192.168.0.2) ⊟ User Datagram Protocol, Src Port: domain (53), Dst Port: 1972 (1972) ⊟ Domain Name System (response) Transaction ID: 0x0001 Flags: 0x8183 (Standard query response, No such name) Questions: 1 Answer RRs: 0 Authority RRs: 1 Additional RRs: 0 Queries ⊟ 1.0.92.157.in-addr.arpa: type PTR, class IN Name: 1.0.92.157.in-addr.arpa Type: PTR (Domain name pointer) Class: IN (0x0001) ⊟ Authoritative nameservers ⊟ 0.92.157.in-addr.arpa: type SOA, class IN, mname ns1.uba.ar Name: 0.92.157.in-addr.arpa Type: SOA (Start of zone of authority) Class: IN (0x0001) Time to live: 3 hours Data length: 43 Primary name server: ns1.uba.ar Responsible authority's mailbox: oper.ccc.uba.ar Serial number: 2007071600 Refresh interval: 10 hours Retry interval: 1 hour Expiration limit: 5 days Minimum TTL: 10 hours </pre>					
<pre> 0020 00 02 00 35 07 b4 00 68 4c c1 00 01 81 83 00 01 ...5...h L... 0030 00 00 00 01 00 00 01 31 01 30 02 39 32 03 31 351 .0.92.15 0040 37 07 69 6e 2d 61 64 64 72 04 61 72 70 61 00 00 7.in-add r.arpa.. 0050 0c 00 01 c0 0e 00 06 00 01 00 00 2a 30 00 2b 03*0.+ 0060 6e 73 31 03 75 62 61 02 61 72 00 04 6f 70 65 72 ns1.uba. ar..oper 0070 02 62 62 62 62 62 62 62 7b 70 00 00 8c 30 00 00 ccc.uba. fn </pre>					

Cuarta implementación: Aplicación Cliente-Servidor, programa de Chat.

Finalmente nos decidimos a realizar una aplicación cliente y otra servidor para que se comuniquen entre ellas dentro de una LAN para lograr un programa de mensaje instantáneo.

Primero veamos un esquema del funcionamiento de la parte cliente y servidor.



El servidor realiza entonces las siguientes tareas:

1. Crea un socket de comunicación.
2. Asocia dirección IP y Puerto de conexión al socket.
3. Establece la cola de conexiones entrantes, para atender según orden de llegada.
4. Atiende las conexiones entrantes.
5. Envía y recibe datos.
6. Cierra el socket para terminar la comunicación.

Y el cliente cliente:

1. Crea un socket de comunicación.
2. Se conecta al servidor que esta esperando conexiones entrantes.
3. Envía y recibe datos.
4. Cierra el socket para terminar la comunicación

Las funciones para realizar todas estas tareas fueron especificadas en la introducción a Winsock API.

Código fuente servidor:

```
servidor.c
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock.h>
#define PUERTO 6000 // Puerto de conexion
#define IP "192.168.0.2"

int main()
{
    WORD winsock; // Version del winsock
    WSADATA datos; // Datos para inicializar winsock
    SOCKET escucha; // Con este socket aceptamos conexiones entrantes
    SOCKET comunicacion; // Con este socket enviamos y recibimos datos
    struct sockaddr_in servidor; // Datos para la conexion
    int tam_sockaddr=sizeof(struct sockaddr); // Bytes de la estructura sockaddr
    char carac_enviado[50]="";
    char carac_recibido[50]="";

    // Inicializamos winsock
    winsock=MAKEWORD(1,1);
    if(WSAStartup(winsock, &datos))
        printf("Error: Problemas al inicializar winsock \n");
    else
    {
        // Creamos socket.
        escucha=socket(AF_INET, SOCK_STREAM, 0);
        if(escucha==-1)
            printf("Error: Problemas al crear el socket \n");
        else
        {
            // Cubrimos datos estructura sockaddr_in
            servidor.sin_family=AF_INET;
            servidor.sin_port=htons(PUERTO); // El puerto estará en Formato de Red
            servidor.sin_addr.s_addr=inet_addr(IP); // La ip estará en formato long

            // Asociamos ip y puerto al socket
            if((bind(escucha, (struct sockaddr *)&servidor, sizeof(servidor)))==-1)
                printf("Error: Problemas al asociar puerto e ip al socket \n");
            else
            {
                // Establecemos cola de clientes a 1
                listen(escucha, 1);

                // Aceptamos conexiones entrantes
                printf("Esperando conexiones entrantes... \n");
                comunicacion=accept(escucha, (struct sockaddr *)&servidor, &tam_sockaddr);

                // Como no vamos a aceptar más conexiones cerramos el socket escucha
                close(escucha);
            }
        }
    }
}
```

```

        while(((strcmp("shutdown\n",carac_recibido))&(strcmp("shutdown\n",carac_enviado)))!=0)
        {
            recv (comunicacion, (void *)carac_recibido, 50, 0);
            printf("\nCliente> %s \n", carac_recibido);
            printf("Servidor> ");
            fgets(carac_enviado, 50, stdin);
            send (comunicacion, (const void *)carac_enviado, 50, 0);
        }

        // Cerramos el socket de la comunicacion
        close(comunicacion);
    }
}
// Cerramos liberia winsock
WSACleanup();
}
system("PAUSE");
return (EXIT_SUCCESS);
}

```

Podemos ver como detalle que el servidor necesita abrir 2 sockets, uno para escuchar las conexiones entrantes y el otro para efectuar el envío y la recepción de datos.

Código Fuente Cliente:

```

[*] cliente.c
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winsock.h>
#define IP "192.168.0.2"
#define PUERTO 6000 // Puerto de conexión

int main(int argc, char *argv[])
{
    WORD winsock; //Version del winsock
    WSADATA datos; // Datos Para inicializar el Winsock
    SOCKET comunicacion; // socket de envio y recepcion de datos
    struct sockaddr_in servidor; //Datos para la conexión
    int estado_conexion;
    int num_enviado=2;
    int num_recibido=4;
    char carac_recibido[50]="";
    char carac_enviado[50]="";

    fflush(stdin);
    // inicializacion del Winsock
    winsock=MAKEWORD(1,1);

    if(WSAStartup(winsock, &datos))
        printf("Error: Problemas al inicializar winsock \n");
    else
    {
        comunicacion=socket(AF_INET, SOCK_STREAM, 0); //Creamos socket de datos
        if(comunicacion== -1)
            printf("Error: Problemas al crear el socket \n");
        else

```

```
[*] cliente.c
comunicacion=socket(AF_INET, SOCK_STREAM, 0); //Creamos socket de datos
if(comunicacion==-1)
    printf("Error: Problemas al crear el socket \n");
else
{
    // Cubrimos datos de la estructura sockaddr_in
    servidor.sin_family=AF_INET;
    servidor.sin_port=htons(PUERTO); //el puerto estara en formato de Red
    servidor.sin_addr.s_addr=(inet_addr(IP)); // la ip en formato long

    //nos conectamos al servidor

    estado_conexion=connect(comunicacion, (struct sockaddr *)&servidor, sizeof(servidor));
    if(estado_conexion==-1)
        printf("Error: No se puede conectar a \"%s\" \n", IP);

    else
    {
        //inicio de rutina de envio y recepcion de datos

        while(((strcmp("shutdown\n",carac_recibido))&(strcmp("shutdown\n",carac_enviado)))!=0)
        {

            printf("Cliente> ");
            fgets(carac_enviado,50,stdin);
            send(comunicacion, (const void *)carac_enviado, 50, 0);
            recv(comunicacion, (void *)carac_recibido, 50, 0);
            printf("Servidor> %s \n", carac_recibido);

        }

    }

}

close(comunicacion); // cierro el socket

}

WSACleanup(); // cierro winsock
}

system("PAUSE");
return (EXIT_SUCCESS);
}
```

```
[*] cliente.c

else
{
    //inicio de rutina de envio y recepcion de datos

    while(((strcmp("shutdown\n",carac_recibido))&(strcmp("shutdown\n",carac_enviado)))!=0)
    {

        printf("Cliente> ");
        fgets(carac_enviado,50,stdin);
        send(comunicacion, (const void *)carac_enviado, 50, 0);
        recv(comunicacion, (void *)carac_recibido, 50, 0);
        printf("Servidor> %s \n", carac_recibido);

    }

}

}

WSACleanup(); // cierro winsock
}

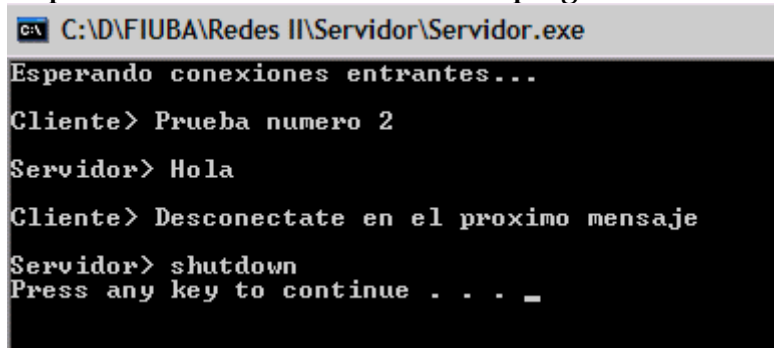
system("PAUSE");
return (EXIT_SUCCESS);
}
```

La diferencia con el programa servidor, es que el cliente solo necesita un socket mediante el cual enviar y recibir los datos. Pero no necesita un socket para escuchar las conexiones entrantes, ya que no las tiene.

El programa funciona como un "handy" es decir, se envía un mensaje del cliente al servidor y luego del servidor al cliente hasta que uno de los dos corta la comunicación enviando un "shutdown".

Procedimos luego a comprobar el funcionamiento del aplicativo, realizando las siguientes capturas de pantalla.

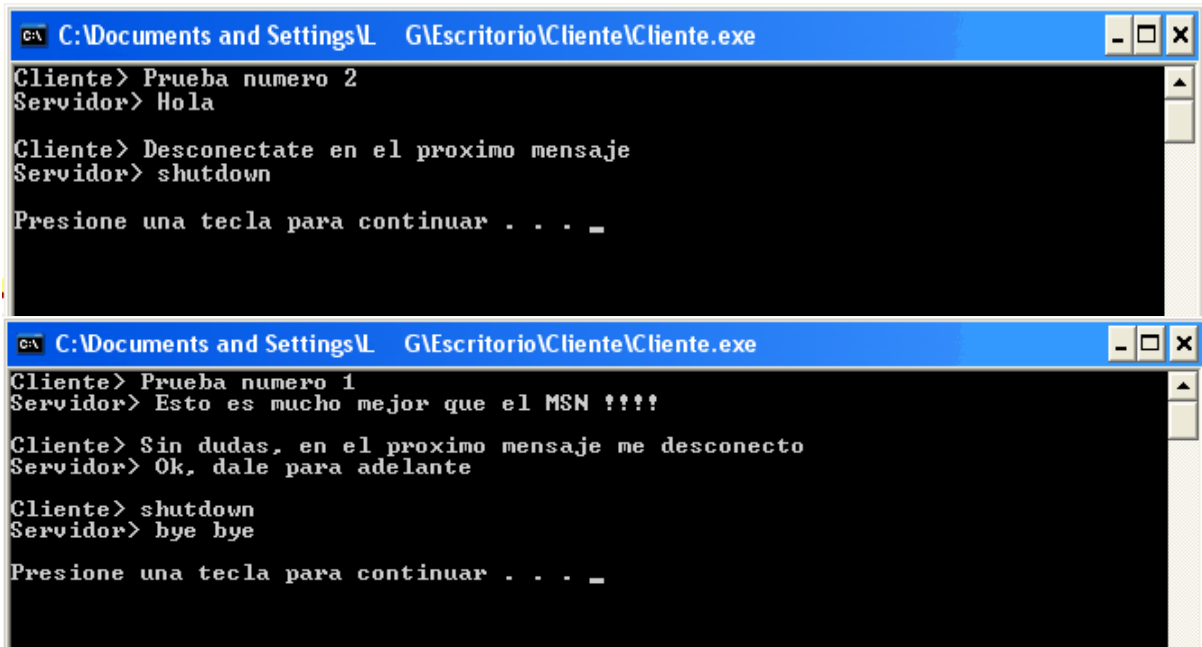
Primero probamos el funcionamiento del programa servidor



```
C:\D\FIUBA\Redes II\Servidor\Servidor.exe
Esperando conexiones entrantes...
Cliente> Prueba numero 2
Servidor> Hola
Cliente> Desconectate en el proximo mensaje
Servidor> shutdown
Press any key to continue . . . _
```

Se ve lo explicado anteriormente, se envía de a 1 mensaje por vez y se termina cuando alguno de los dos escribe “shutdown”.

Ahora veamos el lado cliente



```
C:\Documents and Settings\G\Escritorio\Ciente\Ciente.exe
Cliente> Prueba numero 2
Servidor> Hola
Cliente> Desconectate en el proximo mensaje
Servidor> shutdown
Presione una tecla para continuar . . . _

C:\Documents and Settings\G\Escritorio\Ciente\Ciente.exe
Cliente> Prueba numero 1
Servidor> Esto es mucho mejor que el MSN !!!!
Cliente> Sin dudas, en el proximo mensaje me desconecto
Servidor> Ok, dale para adelante
Cliente> shutdown
Servidor> bye bye
Presione una tecla para continuar . . . _
```

En la primer captura de pantalla podemos ver la desconexión del lado servidor en el cliente y luego una desconexión del cliente del lado del cliente.

Podemos ver un sniffing de la interfase de conexión donde se aprecia el 3-way handshaking asi como la desconexión de parte del cliente.

Filter: `ip.addr == 192.168.0.1` Expression... Clear Apply

No. -	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [SYN] Seq=0 Len=0 MSS=1460
2	0.000118	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [SYN, ACK] Seq=0 Ack=1 win=65535 Len=0 MSS=1460
3	0.000346	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [ACK] Seq=1 Ack=1 win=64240 Len=0
4	12.655321	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
5	12.806647	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [ACK] Seq=1 Ack=51 win=65485 Len=0
14	34.456335	192.168.0.2	192.168.0.1	X11	Event: <Unknown eventcode 69>
15	34.643212	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [ACK] Seq=51 Ack=51 win=64190 Len=0
16	75.930255	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
17	76.097659	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [ACK] Seq=51 Ack=101 win=65435 Len=0
18	87.034335	192.168.0.2	192.168.0.1	X11	Event: ClientMessage, <Unknown eventcode 97>
19	87.148432	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [ACK] Seq=101 Ack=101 win=64140 Len=0
20	96.741129	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
21	96.927629	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [ACK] Seq=101 Ack=151 win=65385 Len=0
22	102.009700	192.168.0.2	192.168.0.1	X11	Error: Success
23	102.009814	192.168.0.2	192.168.0.1	TCP	6000 > 3497 [RST, ACK] Seq=151 Ack=151 win=0 Len=0

Ethernet II, Src: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e), Dst: Toshiba_a8:37:d2 (00:08:0d:a8:37:d2)
 Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
 Transmission Control Protocol, Src Port: 3497 (3497), Dst Port: 6000 (6000), Seq: 0, Len: 0
 Source port: 3497 (3497)
 Destination port: 6000 (6000)
 Sequence number: 0 (relative sequence number)
 Header length: 28 bytes
 Flags: 0x0002 (SYN)
 0... .. = Congestion window Reduced (CWR): Not set
 .0.. .. = ECN-Echo: Not set
 ..0. = Urgent: Not set
 ...0 = Acknowledgment: Not set
 0... = Push: Not set
0.. = Reset: Not set
1. = Syn: Set
0 = Fin: Not set
 Window size: 64240
 Checksum: 0xebbf [correct]
 Options: (8 bytes)
 Maximum segment size: 1460 bytes

Y ahora con una prueba similar desde el lado servidor

Filter: Expression... Clear Apply

No. -	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.1	192.168.0.2	TCP	3503 > 6000 [SYN] Seq=0 Len=0 MSS=1460
2	0.000131	192.168.0.2	192.168.0.1	TCP	6000 > 3503 [SYN, ACK] Seq=0 Ack=1 win=65535 Len=0 MSS=1460
3	0.000363	192.168.0.1	192.168.0.2	TCP	3503 > 6000 [ACK] Seq=1 Ack=1 win=64240 Len=0
4	9.555421	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
5	9.694690	192.168.0.2	192.168.0.1	TCP	6000 > 3503 [ACK] Seq=1 Ack=51 win=65485 Len=0
6	12.834516	192.168.0.2	192.168.0.1	X11	Event: <unknown eventcode 72>
7	13.020755	192.168.0.1	192.168.0.2	TCP	3503 > 6000 [ACK] Seq=51 Ack=51 win=64190 Len=0
8	27.059661	192.168.0.1	192.168.0.2	TCP	[TCP segment of a reassembled PDU]
9	27.219895	192.168.0.2	192.168.0.1	TCP	6000 > 3503 [ACK] Seq=51 Ack=101 win=65435 Len=0
10	32.841730	192.168.0.2	192.168.0.1	X11	Error: Success, Success
11	32.841848	192.168.0.2	192.168.0.1	TCP	6000 > 3503 [RST, ACK] Seq=101 Ack=101 win=0 Len=0

Frame 1 (62 bytes on wire, 62 bytes captured)
 Ethernet II, Src: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e), Dst: Toshiba_a8:37:d2 (00:08:0d:a8:37:d2)
 Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.2 (192.168.0.2)
 Transmission Control Protocol, Src Port: 3503 (3503), Dst Port: 6000 (6000), Seq: 0, Len: 0
 Source port: 3503 (3503)
 Destination port: 6000 (6000)
 Sequence number: 0 (relative sequence number)
 Header length: 28 bytes
 Flags: 0x0002 (SYN)
 0... .. = Congestion window Reduced (CWR): Not set
 .0.. .. = ECN-Echo: Not set
 ..0. = Urgent: Not set
 ...0 = Acknowledgment: Not set
 0... = Push: Not set
0.. = Reset: Not set
1. = Syn: Set
0 = Fin: Not set
 Window size: 64240
 Checksum: 0xbd21 [correct]
 Options: (8 bytes)
 Maximum segment size: 1460 bytes
 NOP
 NOP
 SACK permitted

Finalmente como corolario podemos ver un paquete de texto enviado desde el cliente al servidor, mostrando que los mensajes iban encapsulados en modo texto.

Filter: `(ip.addr == 192.168.0.1) && (ip.addr == 192.168.0.2)` Expression... Clear Apply

No. .	Time	Source	Destination	Protocol	Info
257	239.728524	192.168.0.2	192.168.0.1	X11	Event: <Unknown eventcode 69>
258	239.915274	192.168.0.1	192.168.0.2	TCP	3497 > 6000 [ACK] Seq=51 Ack=51 win=64190 Len=0

⊕ Frame 257 (104 bytes on wire, 104 bytes captured)
 ⊕ Ethernet II, Src: Toshiba_a8:37:d2 (00:08:0d:a8:37:d2), Dst: Asiarock_44:ce:4e (00:0b:6a:44:ce:4e)
 ⊕ Internet Protocol, Src: 192.168.0.2 (192.168.0.2), Dst: 192.168.0.1 (192.168.0.1)
 ⊖ Transmission Control Protocol, Src Port: 6000 (6000), Dst Port: 3497 (3497), Seq: 1, Ack: 51, Len: 50
 Source port: 6000 (6000)
 Destination port: 3497 (3497)
 Sequence number: 1 (relative sequence number)
 [Next sequence number: 51 (relative sequence number)]
 Acknowledgement number: 51 (relative ack number)
 Header length: 20 bytes
 ⊕ Flags: 0x0018 (PSH, ACK)
 window size: 65485
 Checksum: 0xb543 [correct]
 TCP segment data (18 bytes)
 = X11, Event, eventcode: 69 (<Unknown eventcode 69>)
 eventcode: 69 (<Unknown eventcode 69>)
 undecoded

```

0000 00 0b 6a 44 ce 4e 00 08 0d a8 37 d2 08 00 45 00  ..}D.N.. ..7...E.
0010 00 5a 4d 8d 40 00 80 06 2b bd c0 a8 00 02 c0 a8  .ZM.@... +.....
0020 00 01 17 70 0d a9 93 cb e0 d5 9b 17 5b 1d 50 18  ...p... ..[.P.
0030 ff cd b5 43 00 00 45 73 74 6f 20 65 73 20 6d 75  ...C...Es to es mu
0040 63 68 6f 20 6d 65 6a 6f 72 20 71 75 65 20 65 6c  cho mejo r que el
0050 20 4d 53 4e 20 21 21 21 21 0a 00 00 00 00 00 00  MSN !!! !.....
0060 00 00 00 00 00 00 00 00  .....
```