

Programación de red



Conceptos fundamentales

- Ancho de banda
- Latencia
- Nodos



Conceptos de sockets

- Socket → Permite el intercambio de cualquier flujo de datos entre dos aplicaciones remotas.
 - Actúa como un “túnel” entre 2 procesos de dos equipos, a través del cual intercambiarán datos.
 - Un *socket* está definido por una dupla formada por la IP del equipo y su Puerto. Además cada socket es único.



Conceptos de sockets

■ Tipos de sockets

☐ Socket TCP (Stream)

- SOCK_STREAM
- Orientado a conexión
- Confiable
- Garantizado el orden, NO duplicidad de datos

☐ Socket UDP (Datagram)

- SOCK_DGRAM
- No orientado a conexión
- No confiable
- No está garantizado el orden, puede haber duplicidad de datos



Conceptos de sockets

■ Dominios de un socket

- Permiten indicar el formato de las direcciones que utilizarán los sockets
- Hay dos tipos:
 - AF_UNIX: Para conexiones en los que el cliente y servidor están en la misma máquina. Permite la comunicación entre dos procesos locales.
 - AF_INET: Cliente y servidor están en cualquier máquina.
 - La comunicación se basa en el conjunto de protocolos TCP/IP.
 - La identificación del socket se basa en IP:PUERTO



Conceptos de sockets

■ Arquitectura

□ Cliente / Servidor

■ Servidor Concurrente.

- Disponemos de 2 sockets:
 - Socket de escucha (Listening)
 - Socket dinámico que maneja la conexión con el cliente.
- El socket de escucha, estará a la espera de nuevos clientes y creará nuevos sockets para todas las nuevas conexiones.

■ Servidor Iterativo.

- Disponemos de un único socket.
- Conexión 1 a 1



Conceptos de sockets

■ Distintas implementaciones

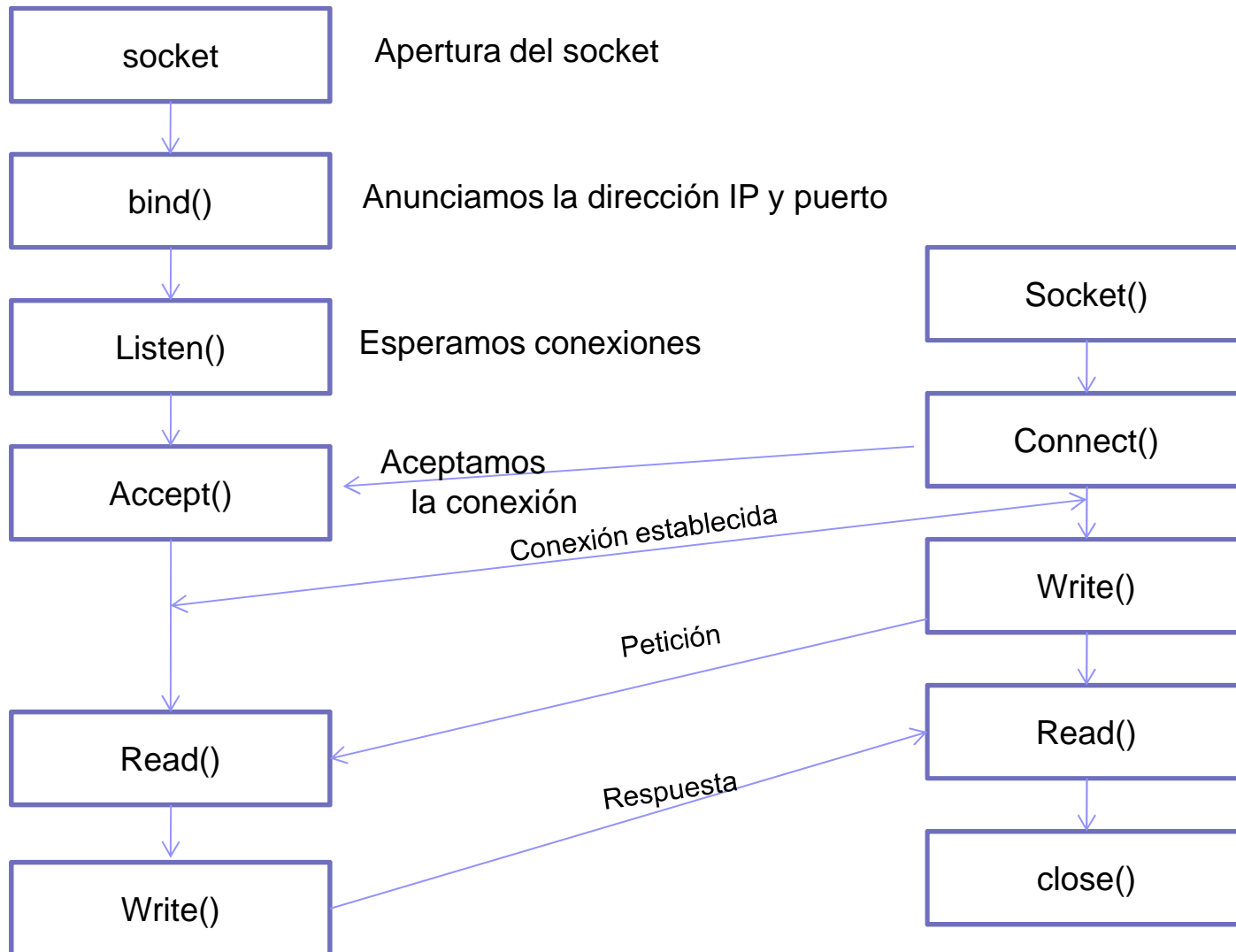
☐ Linux / Unix

- Sockets API
- BSD sockets

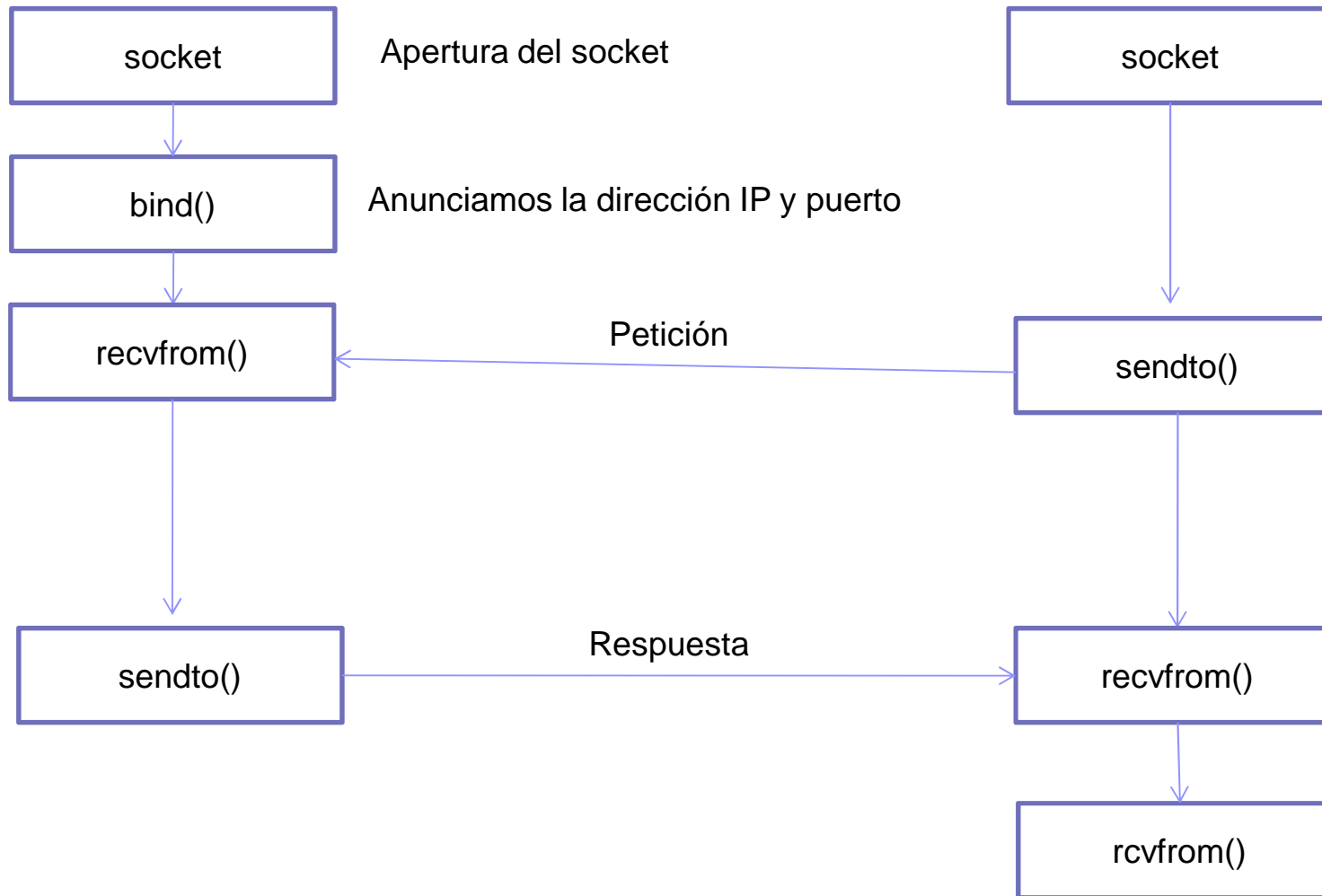
☐ Windows

- Winsock
- Winsock 2

Socket SOCK_STREAM: TCP

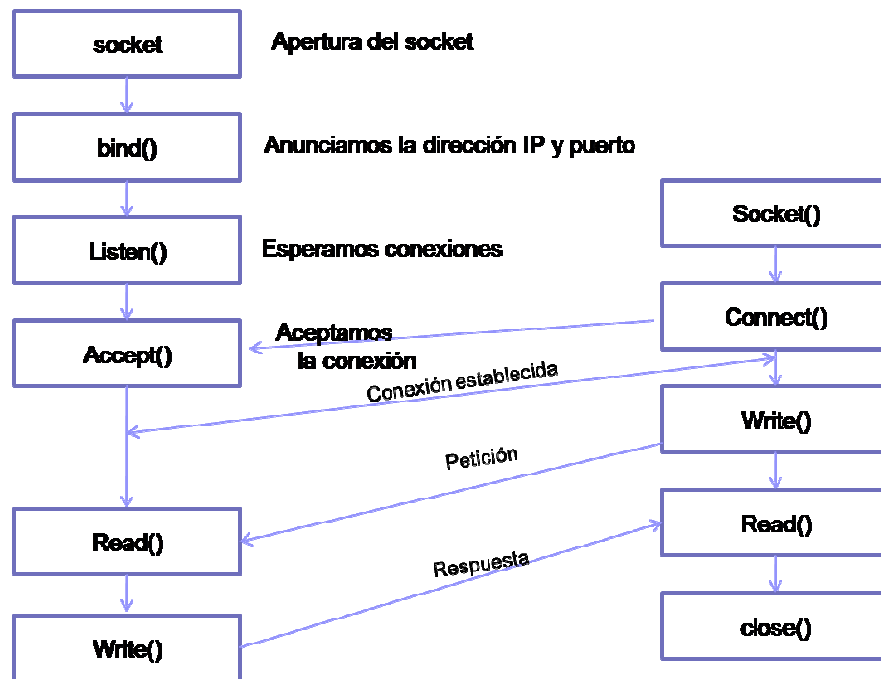


Socket SOCK_DGRAM: UDP

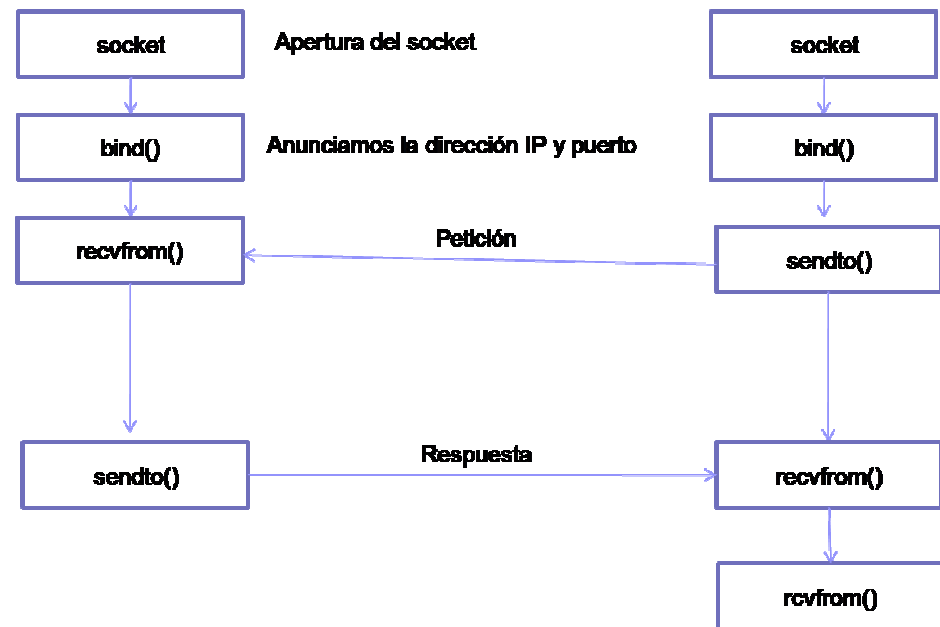


TCP vs. UDP

SOCK_STREAM



SOCK_DGRAM



Inicializar winsock : **SÓLO WINDOWS!**

- Librería **winsock.h** || **winsock2.h**
- Obligatorio en sistemas Windows.
- Definiremos una variable de tipo **WSADATA** → **WinSockApi**
 - WSADATA wsa;

C++

```
typedef struct WSADATA {  
    WORD          wVersion;  
    WORD          wHighVersion;  
    char          szDescription[WSADESCRIPTION_LEN+1];  
    char          szSystemStatus[WSASYS_STATUS_LEN+1];  
    unsigned short iMaxSockets;  
    unsigned short iMaxUdpDg;  
    char FAR *     lpVendorInfo;  
} WSADATA, *LPWSADATA;
```



Inicializar winsock : **SÓLO WINDOWS!**

- Posteriormente, inicializamos la variable WSADATA, con la función **WSAStartup()**.

- int WSAStartup(__in WORD wVersionRequested, __out LPWSADATA lpWSAData)

- WSAStartup(MAKEWORD(2,0),&wsa);
 - Devuelve **0** si todo ha ido **correctamente**.
 - Errores devueltos a través de la función WSAGetLastError():

- WSASYSNOTREADY → Subsistema de red no listo
 - WSAEFAULT → Estructura WSAData no válida
 - WSAVERNOTSUPPORTED → Versión no soportada



Liberación WSA: **SÓLO WINDOWS!**

- Para poder liberar la DLL de winsock inicializada con WSAStartup(), haremos uso de la función **WSACleanup()**
 - int WSACleanup(void)
 - Devuelve **0** si todo ha ido **correctamente**.
 - Errores devueltos a través de la función WSAGetLastError():
 - WSA_NOTINITIALIZED → No se ha invocado, previamente, el WSAStartup.
 - WSAENETDOWN → El subsistema de red ha fallado.



Inicializar winsock : **SÓLO WINDOWS!**

```
#include "stdio.h"  
#include "winsock.h"
```

```
int main()  
{  
    WSADATA wsa;  
    int i;  
    i=WSAStartup(MAKEWORD(2,0), &wsa);  
    if(i)  
    {  
        printf("Error:%s",WSAGetLastError());  
        return 1;  
    }  
  
    if(LOBYTE(wsa.wVersion)!=2)  
    {  
        printf("La versión 2 de Winsock no está instalada en este equipo!");  
        WSACleanup();  
        return 1;  
    }  
    printf("Todo correcto");  
  
}
```



Creación del Socket

- Definimos la familia de dirección (Capa 3) y el tipo de información que transmitirá en capa 4 (Protocolo L4). → Creamos la conexión
- Devuelve un identificador de SOCKET con las características indicadas en el constructor.
- `SOCKET socket (int af, int type, int protocol);`
 - AF → Familia de dirección (AF_INET)
 - Type → Tipo de segmento (SOCK_STREAM | SOCK_DGRAM || SOCK_RAW)
 - Protocol → Protocolo de capa 4 a utilizar.
 - 0 → El sistema elige el protocolo

Familias de Dirección `socket(int af, int, type, int proto)`

Af	Meaning
AF_UNSPEC 0	The address family is unspecified.
AF_INET 2	The Internet Protocol version 4 (IPv4) address family.
AF_IPX 6	The IPX/SPX address family. This address family is only supported if the NWLink IPX/SPX NetBIOS Compatible Transport protocol is installed. This address family is not supported on Windows Vista and later.
AF_APPLETALK 17	The AppleTalk address family. This address family is only supported if the AppleTalk protocol is installed. This address family is not supported on Windows Vista and later.
AF_NETBIOS 17	The NetBIOS address family. This address family is only supported if a Windows Sockets provider for NetBIOS is installed.
AF_INET6 23	The Internet Protocol version 6 (IPv6) address family.
AF_IRDA 26	The Infrared Data Association (IrDA) address family. This address family is only supported if the computer has an infrared port and driver installed.
AF_BTH 32	The Bluetooth address family. This address family is supported on Windows XP with SP2 or later if the computer has a Bluetooth adapter and driver installed.

Tipos de segmentos `socket(int af, int, type, int proto)`

Type	Meaning
SOCK_STREAM 1	A socket type that provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism. This socket type uses the Transmission Control Protocol (TCP) for the Internet address family (AF_INET or AF_INET6).
SOCK_DGRAM 2	A socket type that supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. This socket type uses the User Datagram Protocol (UDP) for the Internet address family (AF_INET or AF_INET6).
SOCK_RAW 3	A socket type that provides a raw socket that allows an application to manipulate the next upper-layer protocol header. To manipulate the IPv4 header, the IP_HDRINCL socket option must be set on the socket. To manipulate the IPv6 header, the IPV6_HDRINCL socket option must be set on the socket.
SOCK_RDM 4	A socket type that provides a reliable message datagram. An example of this type is the Pragmatic General Multicast (PGM) multicast protocol implementation in Windows, often referred to as reliable multicast programming . This type is only supported if the Reliable Multicast Protocol is installed.
SOCK_SEQPACKET 5	A socket type that provides a pseudo-stream packet based on datagrams.



Protocolo `socket(int af, int type, int proto)`

protocol	Meaning
BTHPROTO_RFCOMM 3	The Bluetooth Radio Frequency Communications (Bluetooth RFCOMM) protocol. This is a possible value when the <i>af</i> parameter is AF_BT and the <i>type</i> parameter is SOCK_STREAM. This protocol is supported on Windows XP with SP2 or later.
IPPROTO_TCP 6	The Transmission Control Protocol (TCP). This is a possible value when the <i>af</i> parameter is AF_INET or AF_INET6 and the <i>type</i> parameter is SOCK_STREAM.
IPPROTO_UDP 17	The User Datagram Protocol (UDP). This is a possible value when the <i>af</i> parameter is AF_INET or AF_INET6 and the <i>type</i> parameter is SOCK_DGRAM.
IPPROTO_RM 113	The PGM protocol for reliable multicast. This is a possible value when the <i>af</i> parameter is AF_INET and the <i>type</i> parameter is SOCK_RDM. On the Windows SDK released for Windows Vista and later, this value is also called IPPROTO_PGM. This protocol is only supported if the Reliable Multicast Protocol is installed.

Obtener nombre del equipo

■ Función gethostname

C++

```
int gethostname(  
    _Out_ char *name,  
    _In_  int namelen  
);
```

Parameters

name [out]

A pointer to a buffer that receives the local host name.

namelen [in]

The length, in bytes, of the buffer pointed to by the *name* parameter.



```
#include "stdio.h"
#include "winsock2.h"
int main()
{
    char nombre[100];
    WCHAR nombre2[100];
    WSADATA wsa;
    WSAStartup(MAKEWORD(2,0), &wsa);
    gethostname(nombre, sizeof(nombre));
    printf("\nNombre equipo:%s", nombre);
    GetHostNameW(nombre2, sizeof(nombre2));
    printf("\nNombre equipo:%S\n", nombre2);
    system("pause");
    return 0;
}
```



Estructura `in_addr`

- Formato de las direcciones IPv4.

```
struct in_addr
```

```
{
```

```
    u_long  s_addr; //Dirección IP
```

```
};
```

Estructura **sockaddr**

- Struct sockaddr {
 ushort sa_family;
 char sa_data[14];
};
- Struct sockaddr_in {
 short sin_family;
 short sin_port;
 struct in_addr sin_addr;
 char sin_zero[8];
};
- Válido para IPv4

Genérica para soportar IPv4 e IPv6

Específica IPv4



Resolución DNS

■ Función gethostbyname()

- struct hostent* far gethostbyname (const char *name)
 - name → Puntero a la cadena que contiene el nombre a resolver.
- Si no hay errores devuelve una estructura hostent.
- Si hay errores devuelve un puntero null.
 - WSAHOST_NOT_FOUND → Host no encontrado.

```
typedef struct hostent {  
    char FAR      *h_name;  
    char FAR FAR **h_aliases;  
    short         h_addrtype;  
    short         h_length;  
    char FAR FAR **h_addr_list;  
} HOSTENT, *PHOSTENT, FAR *LPHOSTENT;
```

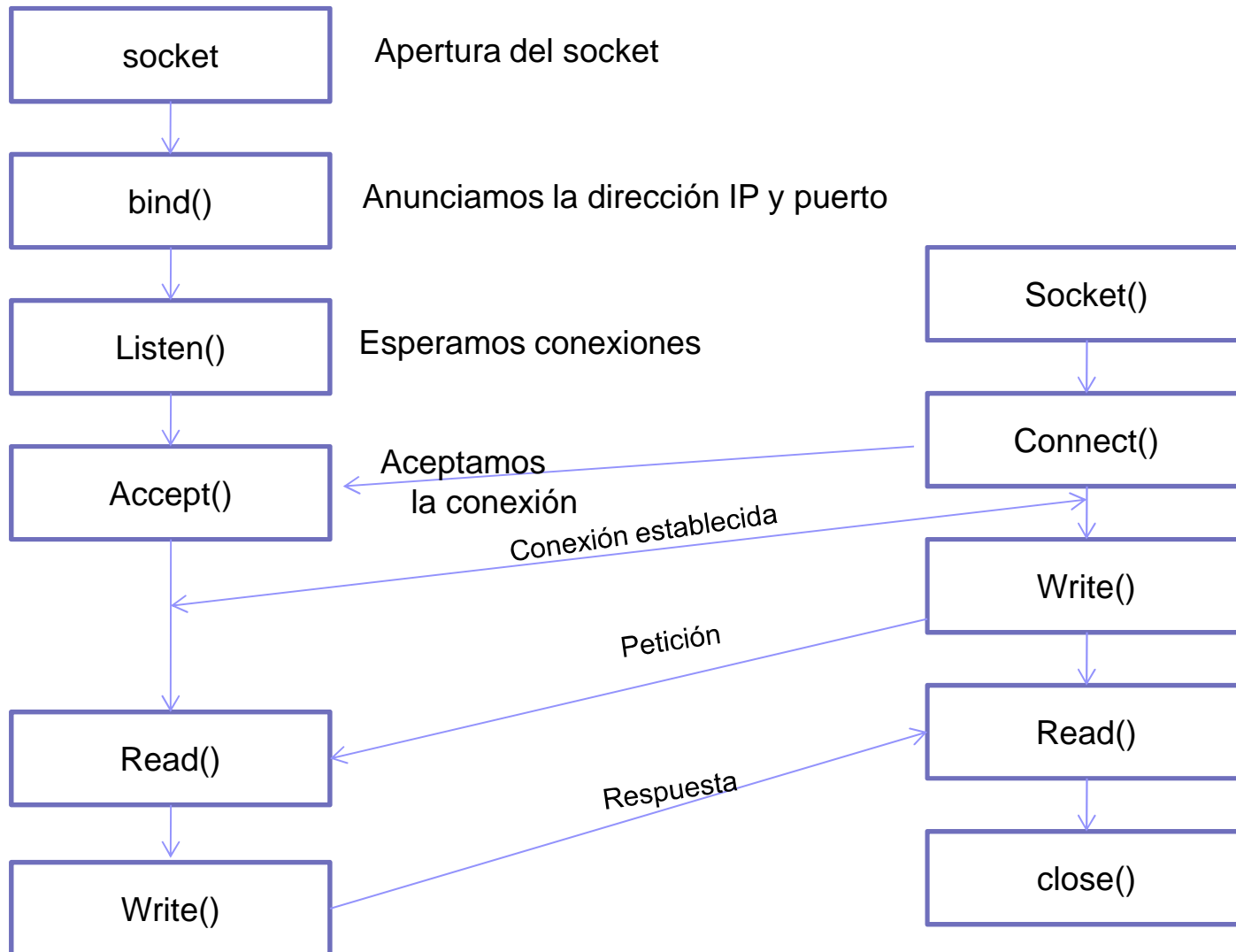


Gethostbyname()

```
#include "stdio.h"
#include "winsock2.h"
int main()
{
    WSADATA wsa;
    SOCKET sock;
    DWORD dwError;
    char ipe[100];
    struct sockaddr_in ip;
    struct hostent *hoste;
    int i=0;
    printf("%d",WSAStartup(MAKEWORD(2, 0),
    &wsa));
    sock = socket(AF_INET, SOCK_STREAM,
    IPPROTO_TCP);
    printf("Introduce nombre equipo:");
    gets(ipe);
    hoste=gethostbyname(ipe);
```

```
    if (hoste == NULL) {
        dwError = WSAGetLastError();
        if (dwError != 0) {
            if (dwError == WSAHOST_NOT_FOUND) {
                printf("Host not found\n");return 1;
            }
            else if (dwError == WSANO_DATA) {
                printf("No data record found\n");
                return 1;
            }
            else {
                printf("Function failed with error: %ld\n", dwError);
                return 1;
            }
        }
    }
    if (hoste->h_addrtype == AF_INET) {
        while (hoste->h_addr_list[i] != 0) {
            ip.sin_addr.s_addr = *(u_long *)hoste->h_addr_list[i++];
            printf("\tDirección IP #%d: %s\n", i, inet_ntoa(ip.sin_addr));
        }
    }
    system("pause");
    return 0;
}
```


Conexiones TCP



Función bind()

- “Unión” entre la IP:Puerto y el Socket.

```
C++  
  
int bind(  
    _In_ SOCKET s,  
    _In_ const struct sockaddr *name,  
    _In_ int namelen  
);
```

- Recibe una estructura de tipo sockaddr!!
- Si todo va bien ==0!!



Abrir un puerto en modo “*Listen*”

- Básico para la IPC.
- Para conseguir que el proceso “escuche” lo que se reciba a través de ese puerto.
- Haremos uso de la función **LISTEN()**

`int listen(SOCKET s, int backlog)`

- SOCKET s → Socket en estado bind().
- Backlog → Número máximo de conexiones. Almacena las peticiones de conexión de los clientes.
 - SOMAXCONN → Máximo definido por el sistema.

Apertura del puerto 27000

```
#include "stdio.h"
#include "winsock2.h"
int main(int argc, char ** argv)
{
    int i;
    WSADATA wsa;
    SOCKET sock;
    struct sockaddr_in ip;
    if(WSAStartup(MAKEWORD(2,0),&wsa))
    {
        printf("Error!");
        return 1;
    }
    sock=socket(AF_INET,
    SOCK_STREAM, IPPROTO_TCP);
    if (sock == INVALID_SOCKET) {
        printf("Error en socket(): %ld\n",
        WSAGetLastError());
        WSACleanup();
        return 1;
    }
}
```

```
ip.sin_family=AF_INET;
ip.sin_addr.s_addr=inet_addr("127.0.0.1");
ip.sin_port=htons(27000);
printf("IP y puerto asignado: %s :
%d",inet_ntoa(ip.sin_addr),ntohs(ip.sin_
port));
if(bind(sock,(SOCKADDR*)&ip,sizeof(ip))==
SOCKET_ERROR)
{
    printf("\nError en bind: %ld\n",
    WSAGetLastError());
    return 1;
}
if(listen(sock, SOMAXCONN))
{
    printf("Error en apertura de puerto");
    return 1;
}
System("pause");
closesocket(sock);
WSACleanup();
return 0;
}
```

¿Cómo aceptamos conexiones entrantes?

- Diferenciación entre TCP y UDP.

- TCP

- Haremos uso de la función **ACCEPT()**

SOCKET accept (SOCKET sock, struct sockaddr *addr, int *addrlen);

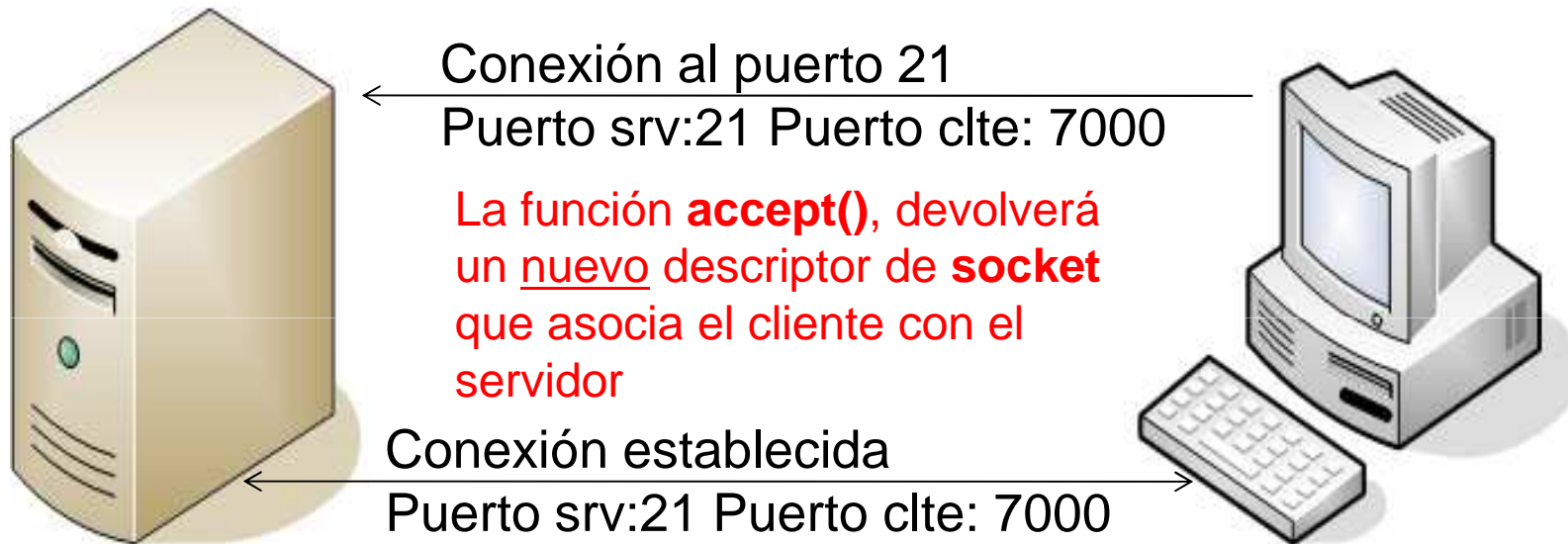
- El socket (sock), debe estar en estado de *listening*.

- El segundo y tercer parámetro, son opcionales y se utilizan para obtener información del cliente (IP | PUERTO)

- La función devuelve un NUEVO descriptor de socket.

- Extrae la primera petición que hay en la cola creada por la función LISTEN();

TCP



Aceptar intento de conexión de un cliente y mostrar su IP y PUERTO TCP

```
#include "stdio.h"
#include "winsock2.h"

int main(int argc, char ** argv)
{
    WSADATA wsa;
    SOCKET sock;
    SOCKET sock_cte;
    struct sockaddr_in ip, ipc;
    if(WSAStartup(MAKEWORD(2,0),&wsa))
    {
        printf("Error");
        return 1;
    }
    //Definimos IP
    ip.sin_family=AF_INET;
    ip.sin_addr.s_addr=inet_addr("0.0.0.0");
    ip.sin_port=htons(7070);

    //Defino Socket
    sock=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

```
if(!sock)
{
    printf("Error al crear el socket");
    return 1;
}

//Ejecuto el BIND
if(bind(sock,(SOCKADDR *)&ip, sizeof(ip)))
{
    printf("Error en BIND");
    return 1;
}

//Abro el puerto!!
if(listen(sock, SOMAXCONN))
{
    printf("Error Apertura puerto");
    return 1;
}
sock_cte=accept(sock, (SOCKADDR *)&ipc, NULL);
printf("Cliente conectado con éxito\n");
printf("IP cliente:%s",inet_ntoa(ipc.sin_addr));
printf("\n Puerto cliente:%d",ntohs(ipc.sin_port));
system("pause");
closesocket(sock);
WSACleanup();
```



¿Conectarse al servidor TCP?

- Permite la conexión entre un cliente TCP y un servidor TCP

- Haremos uso de la función **CONNECT ()**

`int connect(SOCKET sock, struct sockaddr *addr, int addrlen);`

- SOCKET sock → Socket asociado a la nueva conexión.
- Struct sockaddr *addr → IP y PUERTO del servidor.
- int addrlen → Longitud de la estructura sockaddr

- Devuelve un valor tras la conexión *Three-Way handshake*

- 0 → Ok
- SOCKET_ERROR → NOK

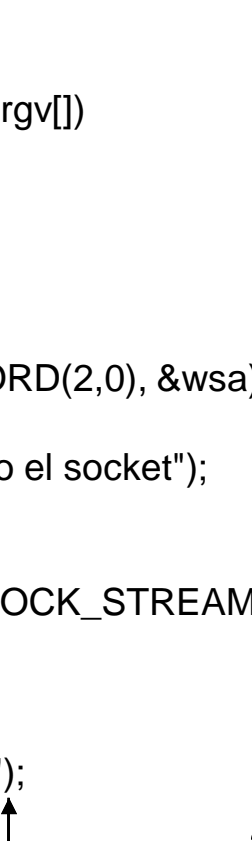
Ejemplo conexión TCP 127.0.0.1:7070

```
#include "stdio.h"
#include "winsock2.h"

int main(int argc, char * argv[])
{
    WSADATA wsa;
    SOCKET sock;
    struct sockaddr_in ip;
    int i;
    if(WSAStartup(MAKEWORD(2,0), &wsa))
    {
        printf("Error inicialiando el socket");
        return 1;
    }
    sock=socket(AF_INET, SOCK_STREAM,
        IPPROTO_TCP);
    if(!sock)
    {
        printf("Error en socket");
        return 1;
    }

    //IP
    ip.sin_family=AF_INET;
    ip.sin_addr.s_addr=inet_addr("127.0.0.1");
    ip.sin_port=htons(7070);

    i=connect(sock, (SOCKADDR *) &ip,
        sizeof(ip));
    if(i==0)
    {
        printf("Conectado correctamente");
    }
    else
    {
        printf("Error en conexión");
    }
    return 0;
}
```





Cerrar conexión TCP

- Cierra sockets TCP.
- Antes de cerrar completamente la conexión, se enviarán todos los datos pendientes en la cola de envío.
- Haremos uso de la función **int CLOSESOCKET(SOCKET sock)**
 - 0 → OK
 - SOCKET_ERROR → NOK



FUNCIONES E/S

TCP



Funciones E/S TCP

- Enviar

- TCP → `send()`

- Socket en estado de *Connected*.

- Recibir

- TCP → `recv()`

- Socket en estado de *Connected*.



Enviar datos: send()

int send(SOCKET s, void *buf, int len, int flags)

- Envía los datos a través del socket s.
- Es OBLIGATORIO que el socket se encuentre en estado de *connected* (*connect()*)
- void *buf → Puntero al buffer que contiene los datos a enviar.
- int len → Bytes a enviar.
- Int flags → Opcionales (0)
 - MSG_DONTROUTE
 - MSG_OOB → Out of Band (Socket SOCK_STREAM)

Ejemplo: Servidor confirma conexión al cliente

```
#include "stdio.h"
#include "winsock2.h"
int main(int argc, char *argv[])
{
    WSADATA wsa;
    SOCKET sock, sock_c;
    struct sockaddr_in ip, ipc;
    char texto[40];
    memset(texto,0,40);
    WSAStartup(MAKEWORD(2,0), &wsa);
    sock=socket(AF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    //ip
    ip.sin_family=AF_INET;
    ip.sin_addr.s_addr=inet_addr("0.0.0.0")
    ;
    ip.sin_port=htons(9999);
```

```
    bind(sock, (SOCKADDR *)&ip,
        sizeof(ip));
    listen(sock, SOMAXCONN);
    sock_c=accept(sock, (SOCKADDR *)
        &ipc, NULL);
    printf("\nCliente %s:%d, conectado con
        éxito", inet_ntoa(ipc.sin_addr),
        ntohs(ipc.sin_port));
    strncpy(texto,"Conectado al sistema",
        40);
    i=send(sock_c, texto, sizeof(texto), 0);
    printf("\nEnviados:%d Bytes",i);
    Return 0
}
```



Recibir datos: recv()

int recv(SOCKET s, void *buf, int len, int flags)

- Similar a la función **send()**
- Es OBLIGATORIO que el socket se encuentre en estado de *connected* (*connect()*)
- void *buf → Puntero al buffer que contendrá los datos recibidos.
- int len → Bytes a recibir.
- Int flags → Opcionales (0)
 - MSG_PEEK → Copia los datos del buffer de entrada pero NO los elimina de dicho buffer.
 - MSG_OOB → Out of Band (Socket SOCK_STREAM)

Ejemplo

Realizar un programa que muestre por pantalla cada uno de los caracteres introducidos por un cliente al puerto 9999

Además, el servidor mostrará un mensaje de bienvenida con la IP del cliente



```
"C:\ESAT\FORMACION\SEGUNDO\TELEM-TICA\2009-2010\Net_Programming\Ejemplo Client_Ser...  
Hola
```

```
C:\> Bienvenido al Sistema Su ip es:172.16.0.  
Hola_
```


Ejemplo

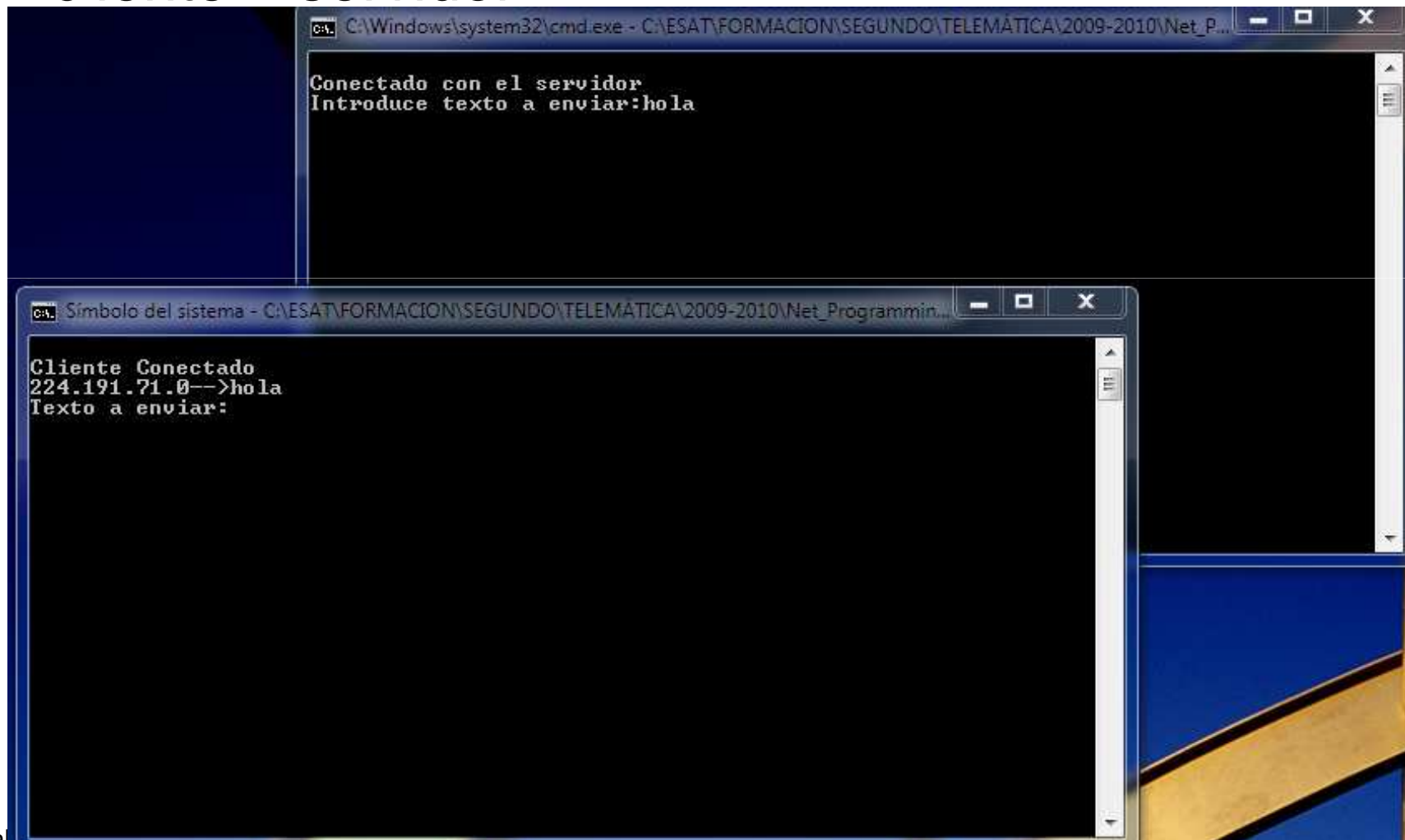
```
#include "stdio.h"
#include "winsock2.h"
int main(int argc, char* argv)
{
    WSADATA wsa;
    SOCKET sock, sock_c;
    struct sockaddr_in ip, ipc;
    char msg_s[1024];
    char msg_c[1024];
    int bytes;
    //Inicializamos WSA y Tipo de Socket
    WSAStartup(MAKEWORD(2,0), &wsa);
    sock=socket(AF_INET, SOCK_STREAM,
        IPPROTO_TCP);
    //IP del server
    ip.sin_family=AF_INET;
    ip.sin_addr.s_addr=inet_addr("0.0.0.0");
    ip.sin_port=htons(9999);
    //Bind
    bind(sock, (SOCKADDR *)&ip, sizeof(ip));
```

```
→ //Ahora a Escuchar
    listen(sock, SOMAXCONN);
    //Espero conexiones
    bytes=sizeof(ipc);
    sock_c=accept(sock,(SOCKADDR *)&ipc,
        &bytes);
    //Inicializo y Envío MSG Bienvenida
    memset(msg_s,0, 1024);
    strcpy(msg_s,"Bienvenido al Sistema\n Su
    ip es:");
    strcat(msg_s,inet_ntoa(ipc.sin_addr));
    strcat(msg_s,"\n");
    send(sock_c,msg_s ,sizeof(msg_s),0);
    do
    {
        bytes=recv(sock_c, msg_c,
            sizeof(msg_c),0);
        //Añado /0
        msg_c[bytes]=0;
        printf("%s",msg_c);
        //printf("Tamaño:%d",bytes);
    }while(bytes>0);

    closesocket(sock_c);
    WSACleanup();
    return 0;
}
```

Ejercicio

- Implementar el un cliente de mensajería básico, cliente – servidor.



The image shows two overlapping Windows command prompt windows. The top window, titled 'C:\Windows\system32\cmd.exe - C:\ESAT\FORMACION\SEGUNDO\TELEMATICA\2009-2010\Net_P...', displays the text 'Conectado con el servidor' and 'Introduce texto a enviar:hola'. The bottom window, titled 'Simbolo del sistema - C:\ESAT\FORMACION\SEGUNDO\TELEMATICA\2009-2010\Net_Programmin...', displays the text 'Cliente Conectado', '224.191.71.0-->hola', and 'Texto a enviar:'.

```
C:\Windows\system32\cmd.exe - C:\ESAT\FORMACION\SEGUNDO\TELEMATICA\2009-2010\Net_P...
Conectado con el servidor
Introduce texto a enviar:hola

Simbolo del sistema - C:\ESAT\FORMACION\SEGUNDO\TELEMATICA\2009-2010\Net_Programmin...
Cliente Conectado
224.191.71.0-->hola
Texto a enviar:
```

Envío de datos

■ Cadenas

- Envío de múltiples campos a través de un carácter delimitador.

- \$cad="campo1:campo2:campo3:campo4

- Troceamos con el strtok(); <string.h>

- *char * strtok (char * str, const char * delimitadores)*

- *Cada llamada a la función devuelve un puntero a la siguiente posición.*

- ```
char str[] = "campo1:campo2:campo";
char * ptr;
ptr = strtok (str, ":");
while (ptr != NULL)
{
 printf("%s\n", ptr);
 ptr = strtok (NULL, ":");
}
```



# Envío de datos

## ■ Estructuras

- Hay que tener en cuenta que la serialización puede dar problemas con las arquitecturas.
- Debemos hacer un “casting” de tipos.

```
struct clientes{
 int id;
 char nombre[50];
 char mapa[50];
 int posx;
 int posy;
};
 clientes cli;
recv(sock_cte,(char *)&cli,sizeof(cli),0);
send(sock_cte,(char *)&cli,sizeof(cli),0);
```

```

Source port: iad3 (1032)
Destination port: 30000 (30000)
[Stream index: 14]
Sequence number: 1 (relative sequence number)
[Next sequence number: 113 (relative sequence number)]
Acknowledgement number: 1 (relative ack number)
Header length: 20 bytes
[-] Flags: 0x18 (PSH, ACK)
 000. = Reserved: Not set
 ...0 = Nonce: Not set
 0... = Congestion window Reduced (CWR): Not set
 0.. = ECN-Echo: Not set
 0. = Urgent: Not set
 1 = Acknowledgement: Set
 1... = Push: Set
 0.. = Reset: Not set
 0. = Syn: Not set
 0 = Fin: Not set
window size value: 64240
[Calculated window size: 64240]
[window size scaling factor: -2 (no window scaling used)]
[+] Checksum: 0xb4bc [validation disabled]
[+] [SEQ/ACK analysis]
[-] Data (112 bytes)
 Data: 0000000057616c74657200cccccccccccccccccccccccc...
 [Length: 112]

```

|      |                                                    |                   |
|------|----------------------------------------------------|-------------------|
| 0000 | 88 53 2e 11 6b f6 88 53 2e 11 6b f6 08 00 45 00    | .S..k..S ..k...E. |
| 0010 | 00 98 00 25 40 00 80 06 97 41 ac 10 0a cb ac 10    | ...%@... .A.....  |
| 0020 | 00 0e 04 08 75 30 59 10 b8 12 b1 1e 14 82 50 18    | ....u0Y. ....P.   |
| 0030 | fa f0 b4 bc 00 00 00 00 00 00 57 61 6c 74 65 72    | ..... ..walter    |
| 0040 | 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc    | .....             |
| 0050 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc    | .....             |
| 0060 | cc cc cc cc cc cc cc cc cc cc cc cc cc 4d 41 50 31 | ..... MAP1        |
| 0070 | 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc    | .....             |
| 0080 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc    | .....             |
| 0090 | cc cc cc cc cc cc cc cc cc cc cc cc cc cc 0a 00    | .....             |
| 00a0 | 00 00 14 00 00 00                                  | .....             |



# Envío de datos

## ■ Información binaria

- ☐ Se envía igual que si fuera char.
- ☐ Hacemos uso de funciones de ficheros para “interpretar” esos “char”.

```
file = fopen(“fichero”, “wb”);
```

```
fwrite(buffer, 1, 512, file) --- fread(buffer, 1, 512, file)
```

```
fseek(website, 0, SEEK_END);
```

```
file_size = ftell(website);
```



# Práctica I

## ■ Servidor Web

- ☐ Protocolo HTTP

- ☐ Cabecera:

HTTP/1.1 200 OK\r\n

Content-Length: 24\r\n

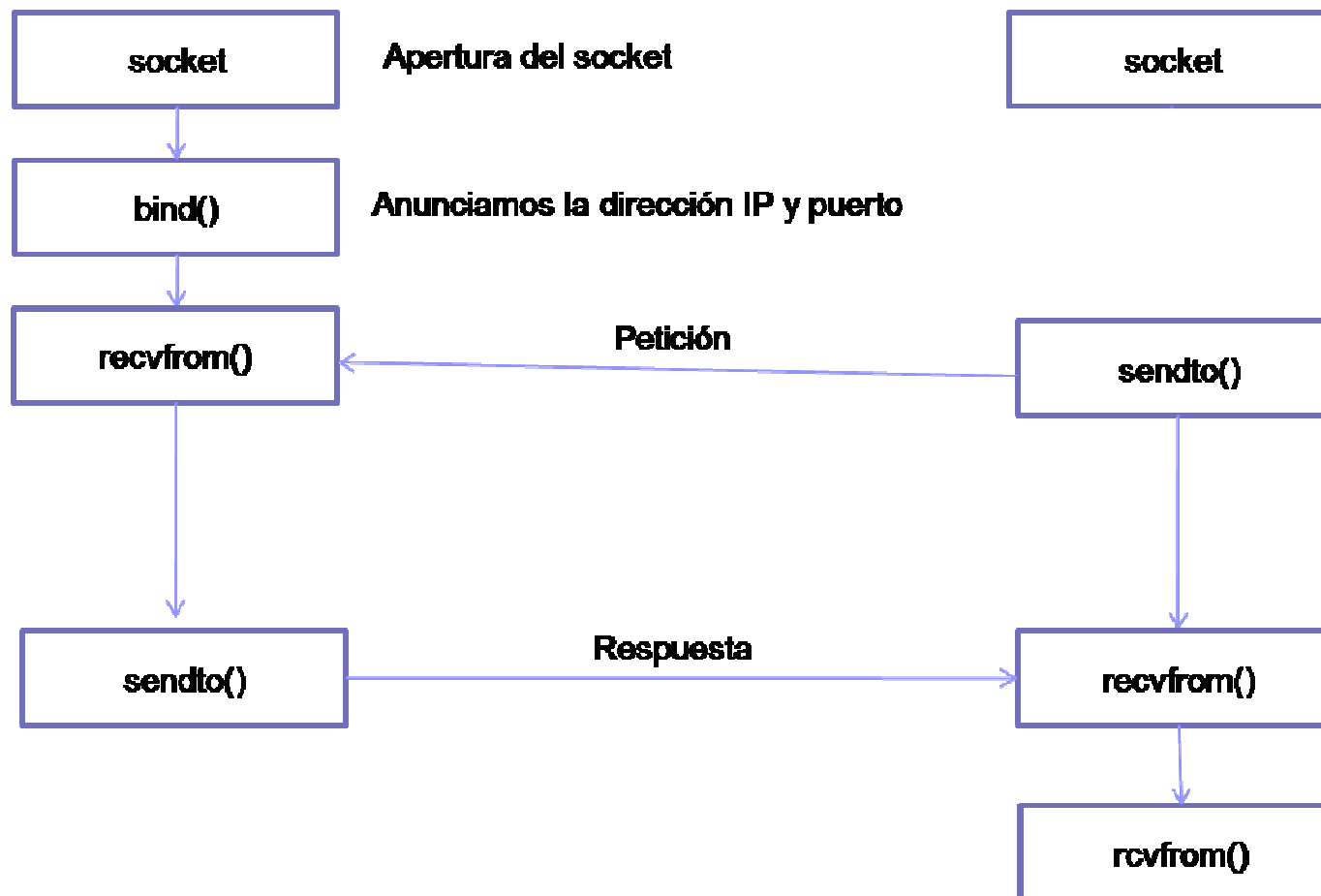
Content-Type: (text/html | image/\* | text/css) \r\n

Server: Apache 2.0.23\r\n

\r\n

[DATA]

# Conexiones UDP







# Enviar datos UDP: sendto()

int sendto(SOCKET s, void \*buf, int len, int flags, struct sockaddr \*to, socklen\_t addrlen)

- Envía los datos a través del socket s al destino *sockaddr \*to (IP + puerto)*.
- NO es necesario que el socket esté en estado *connect()*.
- void \*buf → Puntero al buffer que contiene los datos a enviar.
- int len → Bytes a enviar.
- Int flags → Opcionales (0) ( *Igual que send()* )
  - MSG\_DONTROUTE
  - MSG\_OOB → Out of Band (Socket SOCK\_STREAM)
- struct sockaddr \* to → Destino.
- socklen\_t addrlen → Longitud dirección IP.



# Recibir datos: recvfrom()

int recvfrom(SOCKET s, void \*buf, int len, int flags, struct sockaddr \*from, socklen\_t \*addrlen)

- Similar a la función **sendto()**
- void \*buf → Puntero al buffer que contendrá los datos recibidos.
- int len → Bytes a recibir.
- Int flags → Opcionales (0)
  - MSG\_PEEK → Copia los datos del buffer de entrada pero NO los elimina de dicho buffer.
  - MSG\_OOB → Out of Band (Socket SOCK\_STREAM)
- struct sockaddr \*from → IP y Puerto del host que envía los datos
- socklen\_t \*addrlen → Long. IP



# Ejemplo servidor UDP

```
#include "winsock2.h"
#include "stdio.h"

int main(int argc, char * argv[])
{
 WSADATA wsa;
 int numero, bytes;
 char buff[256];
 int conexion=0;
 char messg[254];
 memset(messg, 0, 254);
 memset(buff, 0, 256);
 struct sockaddr_in ip, ip_c;
 SOCKET sock;
 WSAStartup(MAKEWORD(2,0), &wsa);
 sock=socket(AF_INET, SOCK_DGRAM,
 IPPROTO_UDP);
```

```
//IP + PUERTO
ip.sin_family=AF_INET;
ip.sin_addr.s_addr=inet_addr(argv[1]);
ip.sin_port=htons(atoi(argv[2]));
if(bind(sock, (SOCKADDR *)&ip, sizeof(ip)))
{
 printf("Error de bind()");
 return 1;
}
while(!conexion)
{
 bytes=recvfrom(sock, buff, sizeof(buff), 0, NULL,
 NULL);
 if(bytes>0)
 {
 buff[bytes]=0;
 printf("Datos recibidos:%s, tamaño
 %d", buff, bytes);
 conexion=0;
 }
}
return 0;
}
```



# Ejemplo Cliente UDP

```
#include "sdtio.h"
```

```
.....
```

```
sock=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

```
//IP Server
```

```
ips.sin_family=AF_INET;
```

```
ips.sin_addr.s_addr=inet_addr(argv[1]);
```

```
ips.sin_port=htons(atoi(argv[2]));
```

```
printf("Valor:");
```

```
gets(buff);
```

```
if(sendto(sock, buff, sizeof(buff),0, (SOCKADDR *) &ips,
 sizeof(ips))==SOCKET_ERROR)
```

```
{
```

```
 printf("Error de envio");
```

```
}
```

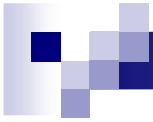
```
else
```

```
{
```

```
 printf("Enviado con éxito");
```

```
}
```

```
return 0;
```



# FUNCIONES E/S

## II



## Funciones E/S (2ª parte)

- ¿Qué pasa si un cliente llama a la función `recv()` y el servidor NO tiene nada que mandar al cliente?
- ¿Cómo podemos detectar eventos? Que nos avise sobre qué sockets tienen datos para leer, cuáles están listos para escribir, y cuáles produjeron excepciones
- ¿Cómo podemos enviar mensajes BROADCAST?



# Función setsockopt()

```
int setsockopt(
 __in SOCKET s,
 __in int level,
 __in int optname,
 __in const char *optval,
 __in int optlen
);
```

- SOCKET s → socket
- Int level → Nivel a definir.
- Int optname → Nombre del valor a definir.
- Char \* optval → Puntero al valor.
- Int optlen → Longitud del valor.



# Función setsockopt()

- Permite definir distintas opciones para el socket indicado como parámetro.
- Nivel SOL\_SOCKET
  - SO\_BROADCAST (BOOL) → Permite el envío de mensajes broadcast.
  - SO\_KEEPALIVE (BOOL) → Habilita los mensajes KEEALIVE en TCP.





# Mensajes Broadcast

- Limitados a una red.
- Socket tipo SOCK\_DGRAM (UDP)
- Dirección 255.255.255.255
- Level → SOL\_SOCKET
- Optname → SO\_BROADCAST

```
setsockopt (sock, SOL_SOCKET, SO_BROADCAST, (char *)&valor,
 sizeof(valor));
```



# Ejemplo: Broadcast

```
#include "stdio.h"
#include "winsock2.h"
int main(int argc, char * argv[])
{
 SOCKET sock;
 WSADATA wsa;
 int valor=1;
 struct sockaddr_in ip;
 char buff[250];
 memset(buff, 0, 250);

 WSStartup(MAKEWORD(2,0), &wsa);
 sock=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
 //IP
 ip.sin_family=AF_INET;
 ip.sin_addr.s_addr=inet_addr("255.255.255.255");
 ip.sin_port=htons(9999);
 printf("Cadena a enviar:");
 gets(buff);
 if(setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (char *)&valor, sizeof(valor))!=SOCKET_ERROR)
 {
 printf("Socket Difusión definido correctamente");
 }
 sendto(sock, buff, sizeof(buff), 0, (SOCKADDR *)&ip, sizeof(ip));
}
```

# Ejemplo: Broadcast

```
+ Frame 360 (554 bytes on wire, 554 bytes captured)
+ Ethernet II, Src: Inventec_ac:83:cc (00:a0:d1:ac:83:cc), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
- Internet Protocol, Src: 192.168.1.234 (192.168.1.234), Dst: 255.255.255.255 (255.255.255.255)
 Version: 4
 Header length: 20 bytes
 + Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
 Total Length: 540
 Identification: 0x3e23 (15907)
 + Flags: 0x00
 Fragment offset: 0
 Time to live: 128
 Protocol: UDP (0x11)
 + Header checksum: 0x381c [correct]
 Source: 192.168.1.234 (192.168.1.234)
 Destination: 255.255.255.255 (255.255.255.255)
- User Datagram Protocol, Src Port: 63440 (63440), Dst Port: distinct (9999)
 Source port: 63440 (63440)
 Destination port: distinct (9999)
 Length: 520
 + Checksum: 0xc4ab [validation disabled]
- Data (512 bytes)
 Data: 686F6C61206120746F646F730000000000000000000000000000...
 [Length: 512]
```



# Ejemplo: WOL

```
#include <stdio.h>
#include <winsock2.h>
int main(int argc, char *argv[])
{
 SOCKET sock;
 WSADATA wsa;
 struct sockaddr_in ip;
 char buffer[102];
 char mac[]={0x01,0x02,0x03,0x04,0x5,0x06};
 int valor=1,contador;
 WSStartup(MAKEWORD(2, 0), &wsa);
 for(contador=0;contador<6;contador++) buffer[contador]=0xff;
 for(contador=6;contador<102;contador+=6) memmove(&buffer[contador],mac,6);
 ip.sin_family=AF_INET;
 ip.sin_addr.s_addr=inet_addr("255.255.255.255");
 ip.sin_port=htons(9);
 sock=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
 valor=setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (char *)&valor,
 sizeof(valor));
 sendto(sock,(char *)buffer,102,0,(SOCKADDR *)&ip, sizeof(ip));
}
```

# Ejemplo WOL

|      |           |               |                 |     |                                                       |
|------|-----------|---------------|-----------------|-----|-------------------------------------------------------|
| 710  | 17.527121 | 192.168.1.234 | 255.255.255.255 | WOL | MagicPacket for woonsang_04:05:06 (01:02:03:04:05:06) |
| 863  | 21.338643 | 192.168.1.234 | 255.255.255.255 | WOL | MagicPacket for woonsang_04:05:06 (01:02:03:04:05:06) |
| 1023 | 25.341071 | 192.168.1.234 | 255.255.255.255 | WOL | MagicPacket for woonsang_04:05:06 (01:02:03:04:05:06) |

|                                                                                               |
|-----------------------------------------------------------------------------------------------|
| Frame 710 (144 bytes on wire, 144 bytes captured)                                             |
| Ethernet II, Src: Inventec_ac:83:cc (00:a0:d1:ac:83:cc), Dst: Broadcast (ff:ff:ff:ff:ff:ff)   |
| Internet Protocol, Src: 192.168.1.234 (192.168.1.234), Dst: 255.255.255.255 (255.255.255.255) |
| Version: 4                                                                                    |
| Header length: 20 bytes                                                                       |
| Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)                           |
| Total Length: 130                                                                             |
| Identification: 0x617a (24954)                                                                |
| Flags: 0x00                                                                                   |
| Fragment offset: 0                                                                            |
| Time to live: 128                                                                             |
| Protocol: UDP (0x11)                                                                          |
| Header checksum: 0x165f [correct]                                                             |
| Source: 192.168.1.234 (192.168.1.234)                                                         |
| Destination: 255.255.255.255 (255.255.255.255)                                                |
| User Datagram Protocol, Src Port: 63490 (63490), Dst Port: tcoregagent (1976)                 |
| Source port: 63490 (63490)                                                                    |
| Destination port: tcoregagent (1976)                                                          |
| Length: 110                                                                                   |
| Checksum: 0xc311 [validation disabled]                                                        |
| Wake On LAN, MAC: woonsang_04:05:06 (01:02:03:04:05:06)                                       |
| Sync stream: FFFFFFFFFF                                                                       |
| MAC: woonsang_04:05:06 (01:02:03:04:05:06)                                                    |

|      |                         |                         |                |
|------|-------------------------|-------------------------|----------------|
| 0000 | ff ff ff ff ff ff 00 a0 | d1 ac 83 cc 08 00 45 00 | .....E.        |
| 0010 | 00 82 61 7a 00 00 80 11 | 16 5f c0 a8 01 ea ff ff | ..az...._..... |
| 0020 | ff ff f8 02 07 b8 00 6e | c3 11 ff ff ff ff ff ff | .....n.....    |
| 0030 | 01 02 03 04 05 06 01 02 | 03 04 05 06 01 02 03 04 | .....          |
| 0040 | 05 06 01 02 03 04 05 06 | 01 02 03 04 05 06 01 02 | .....          |
| 0050 | 03 04 05 06 01 02 03 04 | 05 06 01 02 03 04 05 06 | .....          |
| 0060 | 01 02 03 04 05 06 01 02 | 03 04 05 06 01 02 03 04 | .....          |
| 0070 | 05 06 01 02 03 04 05 06 | 01 02 03 04 05 06 01 02 | .....          |
| 0080 | 03 04 05 06 01 02 03 04 | 05 06 01 02 03 04 05 06 | .....          |



## Ejemplo: KEEP\_ALIVE

- Mantiene la conexión.
- Socket tipo SOCK\_STREAM (TCP)
- Level → SOL\_SOCKET
- Optname → SO\_KEEPALIVE

```
setsockopt (sock, SOL_SOCKET, SO_KEEPALIVE, (char *)&valor,
 sizeof(valor));
```



# Función getsockopt()

```
int getsockopt(
 __in SOCKET s,
 __in int level,
 __in int optname,
 __out char *optval,
 __inout int *optlen
);
```

- Obtiene el valor de un opción del socket.
- SOCKET s → socket
- Int level → Nivel.
- Int optname → Nombre del valor a obtener.
- Char \* optval → Puntero al valor.
- Int \*optlen → Puntero al tamaño.



# Ejemplo: KEEP\_ALIVE

```
BOOL bOptVal = TRUE;
int bOptLen = sizeof(BOOL);
int iOptVal;
int iOptLen = sizeof(int);
if (getsockopt(ListenSocket, SOL_SOCKET, SO_KEEPALIVE, (char*)&iOptVal, &iOptLen) !=
 SOCKET_ERROR)
{
 printf("SO_KEEPALIVE Value: %ld\n", iOptVal);
}
if (setsockopt(ListenSocket, SOL_SOCKET, SO_KEEPALIVE, (char*)&bOptVal, bOptLen) !=
 SOCKET_ERROR)
{
 printf("Set SO_KEEPALIVE: ON\n");
}
if (getsockopt(ListenSocket, SOL_SOCKET, SO_KEEPALIVE, (char*)&iOptVal, &iOptLen) !=
 SOCKET_ERROR)
{
 printf("SO_KEEPALIVE Value: %ld\n", iOptVal);
}
WSACleanup();
return;
```



# Control I/O del socket

## ■ Función **ioctlsocket()**

```
int ioctlsocket(
 __in SOCKET s,
 __in long cmd,
 inout u_long *argp
);
```

- *SOCKET S* → *Descriptor del socket*
- *long cmd* → *Comando a asignar*
- *u\_long \*argp* → *Dirección del parámetro del cmd*
- *Puede ser utilizado en cualquier socket en cualquier estado.*
- *Comandos:*
  - *FIONBIO* → *Modo No bloqueo*
    - *Argumentos: 0* → *Disabled* | *1* → *Enabled*
  - *FIONREAD* → *Cantidad de datos que puedo leer.*
    - *Argumentos* → *Puntero a variable u\_long.*



# Ejemplo:

```
...
WSADATA wsaData;
int iResult = WSASStartup(MAKEWORD(2,2), &wsaData);
if (iResult != NO_ERROR)
 printf("Error at WSASStartup()\n");
SOCKET m_socket; m_socket = socket(AF_INET, SOCK_STREAM,
 IPPROTO_TCP);
if (m_socket == INVALID_SOCKET)
{
 printf("Error at socket(): %ld\n", WSAGetLastError());
 WSACleanup();
 return;
}
//Defino el tipo de socket en modo NO bloqueo
u_long iMode = 1;
ioctlsocket(m_socket, FIONBIO, &iMode);
```