



## Tema 7: Sockets: un interfaz con TCP/IP

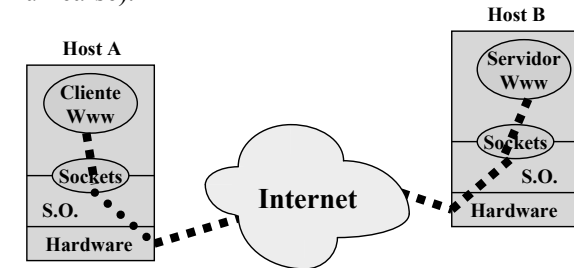
- 1.- Introducción. Conceptos básicos.
  - 1.1. Dirección de un socket.
- 2.- Llamadas básicas de los sockets.
  - 2.1 Diferencias entre BSD-Sockets y WinSockets.
- 3.- Asignación de puertos a procesos.
  - 3.1. Espacio de direcciones de puerto.
- 4.- Atributos de los sockets.

Bibliografía: [COM96] (Cap. 20)



## Las aplicaciones en Internet.

- Utilizan los servicios de TCP/IP.
  - Servicio con conexión: TCP.
  - Servicio sin conexión: UDP.
- Diseño basado en el modelo cliente-servidor.
  - Por cada aplicación tendremos el proceso del cliente y el del servidor (ambos utilizan el interfaz de los sockets para comunicarse).



Tema 7: Sockets: Un interfaz con TCP/IP

2



## Las aplicaciones en Internet.

- El cliente suele ser usado por el usuario, el cual dirige las peticiones de servicio hacia el servidor seleccionado.
  - Ej.: Clientes WWW: NetScape, IExplorer, etc...
- El servidor debe permanecer siempre en marcha esperando las peticiones de servicio de los clientes en un puerto **bien-conocido**.
  - Ej.: Servidores WWW: IIS (NT y W2000), NCSA, etc.

Tema 7: Sockets: Un interfaz con TCP/IP

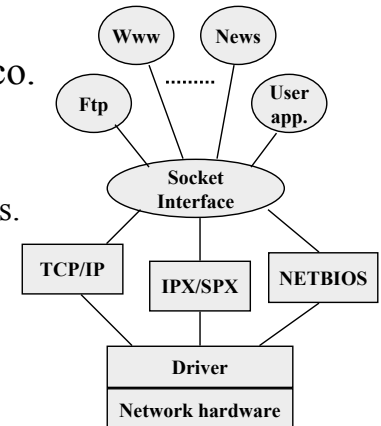
3



## 1. Introducción. Conceptos básicos.

- Orígenes:
  - Introducir TCP/IP en un sistema BSD UNIX (~1980).
  - Necesidad de un interfaz entre las aplicaciones y una arquitectura de red.
- Sockets: Un interfaz genérico.
  - Interfaz estándar.
  - Portabilidad de aplicaciones.
  - Diferentes pilas de protocolos.

Define una librería de primitivas que acceden a los servicios de las pilas de protocolos de un *host*.



Tema 7: Sockets: Un interfaz con TCP/IP

4



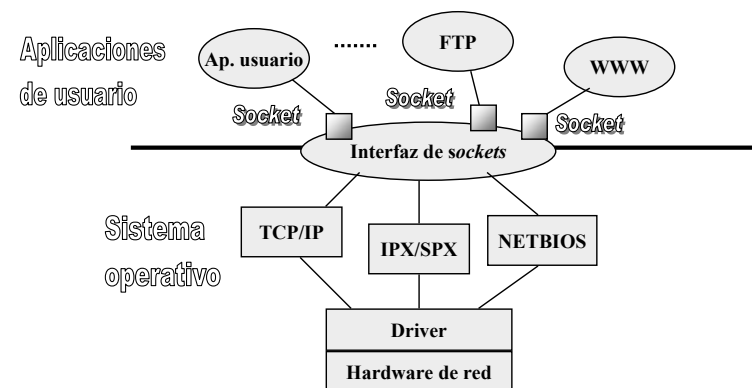
## 1. Introducción. Conceptos básicos.

- Aunque el interfaz de los *sockets* fue escrito inicialmente para máquinas UNIX, hoy en día se ha llevado a otros S.O's (Mac, W9x, OS2, etc.).
- En todos ellos se describe un interfaz “básico” idéntico al original de UNIX\*,
  - Las mismas primitivas con los mismos parámetros.
  - Tipos y estructuras de datos idénticos.
 favoreciendo la portabilidad de aplicaciones.
- En lo que sigue, trabajaremos sobre un sistema UNIX con la pila de protocolos TCP/IP.



## ¿Qué es un socket ?

- Es un punto de acceso (SAP) que una aplicación puede crear para acceder a los servicios de comunicación que ofrecen la pilas de protocolos.



### 1.1 Dirección de un socket.

- Definición:
  - Estructura de datos que permite a las aplicaciones definir una dirección (*endpoint*) única en Internet.
  - Para comunicarse con otras aplicaciones, es necesario definir la dirección del *socket* que se va a utilizar.
- Formato genérico de una dirección de socket.

```
#include <sys/socket.h>

struct sockaddr {
    u_short  sa_family; /* Tipo de dir. */
    char     sa_data[14]; /* valor de la dir. */
}
```



### Dirección de un socket TCP/IP

- Para cada familia de protocolos se define un formato de dirección de *socket* específico. En concreto, para TCP/IP:

```
#include <netinet/in.h>

struct sockaddr_in {
    u_short  sin_family; /* AF_INET */
    u_short  sin_port; /* N° puerto */
    struct in_addr sin_addr; /* Dir. IP */
    char     sin_zero[8];
}

struct in_addr {
    u_long   s_addr; /* Direc. IP */
}
```



## 2. Llamadas básicas de los sockets.

- **SOCKET.**
  - Primitiva que crea un *socket* con el fin de utilizar los servicios de un determinado protocolo de una familia de protocolos residente en el host.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int socket (int family, int type, int protocol);
```

**Family:** AF\_INET, AF\_UNIX, AF\_XNS, AF\_APPLETALK, etc.

**Type:** SOCK\_STREAM (TCP), SOCK\_DGRAM (UDP), SOCK\_RAW (ICMP,IP).

**Protocol:** IP\_PROTO\_TCP, IP\_PROTO\_UDP, IP\_PROTO\_ICMP, IP\_PROTO\_RAW.

Tema 7: Sockets: Un interfaz con TCP/IP

9



## Llamada SOCKET

- La primitiva *socket* devuelve un entero positivo que corresponde al descriptor del *socket* creado.
  - En caso de error, devuelve “-1” y en la variable global “errno” se describe el tipo de error (Común a todas las llamadas del interfaz)
- El descriptor de *socket* que devuelve esta llamada, se utilizará al hacer una operación sobre él (enviar datos, establecer una conexión TCP, etc...), distinguiendolo de otros *sockets*.
- Ejemplo de uso:

```
int sd; /* variable que guardará el descriptor del socket */

sd = socket (AF_INET, SOCK_DGRAM, 0);
```

Tema 7: Sockets: Un interfaz con TCP/IP

10



## Llamada BIND.

- Asigna una dirección local a un *socket* recién creado (llamada *socket*).
- Antes de utilizar un *socket* es necesario asignarle una dirección que lo identifique.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int bind (int sockfd, struct sockaddr* myaddr, int addrlen);
```

**sockfd:** Descriptor de socket al que vamos a asignar una dirección.

**myaddr:** Puntero a una estructura donde se guarda la dirección.

**addrlen:** Tamaño (en bytes) de la dirección de socket (INET = 16 ).

Tema 7: Sockets: Un interfaz con TCP/IP

11



## Llamada BIND.

- Los servidores definirán una dirección de *socket* en la que figure el puerto **bien-conocido**.
- Los clientes pueden especificar uno, aunque normalmente relegan la elección de puerto al sistema (ver ejemplo).
- Ejemplo de uso:

```
int sd, err;
sockaddr_in myaddr; /* variable de tipo dirección de socket INET*/

sd = socket (AF_INET, SOCK_DGRAM, 0);
myaddr.sin_family = AF_INET;
myaddr.sin_addr.s_addr = INADDR_ANY;
myaddr.sin_port = 0;
err = bind (sd, (struct sockaddr *) &myaddr, sizeof(myaddr));
```

Tema 7: Sockets: Un interfaz con TCP/IP

12



## Llamada **CONNECT**.

- Los clientes la utilizan para establecer una conexión (asociación) con un servidor.
  - Devuelve un entero indicando el resultado de la operación.
    - “-1”: Error en el intento de conexión (*errno*).
    - “0”: Conexión establecida.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int connect (int sd, struct sockaddr* destaddr, int addrlen);
```

**sd**: Descriptor de socket sobre el que realizaremos la conexión.

**destaddr**: Dirección de socket del servidor.

**addrlen**: Tamaño (en bytes) de la dirección de socket (INET = 16 ).

Tema 7: Sockets: Un interfaz con TCP/IP

13



## Llamada **CONNECT**.

- Conexión o asociación:

```
{Protocolo, IP origen, Puerto origen, IP destino, Puerto destino}
```

- Características de la llamada **CONNECT**
  - Realiza internamente un **BIND** del *socket*.
  - Obligatoria para clientes **TCP**.
  - Puede usarse con *sockets* de tipo **SOCK\_DGRAM** estableciendo una asociación entre cliente y servidor.
    - UDP** no realiza acción alguna.
    - Tras el **connect**, se pueden enviar los datos sin identificar el destino (asociación establecida).
    - Sólo se aceptarán datagramas de la dirección indicada.

Tema 7: Sockets: Un interfaz con TCP/IP

14



## Llamada **CONNECT**.

- Ejemplo de uso:

```
#define host "158.42.53.99" /* Dir IP del servidor */
#define port 7 /* Puerto donde se encuentra el servidor de ECHO */

int sd, err;
struct sockaddr_in servaddr;

sd = socket (AF_INET, SOCK_STREAM, 0);

servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_aton(host);
servaddr.sin_port = htons(port);

err = connect (sd, (struct sockaddr *) &servaddr, sizeof(servaddr));

if (err < 0) {
    printf ("Can't connect to %s: %s\n", host, sys_errlist[errno]);
    exit (-1);
}
```

Tema 7: Sockets: Un interfaz con TCP/IP

15



## Llamada **LISTEN**.

- Pasa el *socket* a modo pasivo, dispuesto a encolar peticiones de conexión.
  - Se suele usar después de **SOCKET** y **BIND**, y justo antes de **ACCEPT** (servidores).
  - Sólo para *sockets* de tipo **SOCK\_STREAM** (**TCP**).

```
#include <sys/socket.h>
```

```
int listen (int sd, int nconn);
```

**sd**: Descriptor de socket en el que aceptaremos peticiones de conexión.

**nconn**: Tamaño de la cola donde se almacenan las peticiones de conexión entrantes (Normalmente 5).

Tema 7: Sockets: Un interfaz con TCP/IP

16



## Llamada ACCEPT.

- Espera la llegada de una petición de conexión sobre el *socket* previamente inicializado.
  - Sólo para servidores TCP.
  - Se usa tras *SOCKET*, *BIND* y *LISTEN*.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int accept (int sd, struct sockaddr *addr, int *addrlen);
```

**sd:** Descriptor de socket donde esperaremos peticiones de conexión.  
**addr:** Devuelve la dirección del cliente.  
**addrlen:** Tamaño en bytes de la dirección del cliente.

**accept** devuelve un entero, que en caso de éxito, corresponde al descriptor de un nuevo socket asociado a la conexión establecida.



## Características de la llamada ACCEPT.

- Cuando se invoca, bloquea el programa hasta que llegue una petición de conexión sobre el *socket*.
- Si se tienen peticiones encoladas (*LISTEN*) intenta establecer una conexión con la primera.
- El *socket* que se pasa como argumento (*socket* maestro) no tiene definida una asociación, mientras que el nuevo *socket* que devuelve *ACCEPT* si que tiene definida la asociación entre ambos procesos.

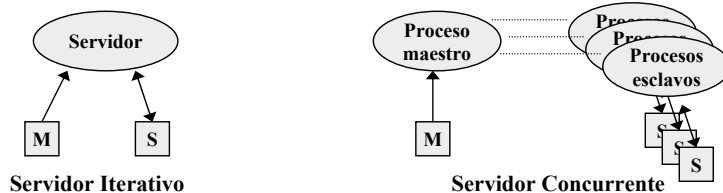
Socket maestro: { TCP, IP fuente, Puerto fuente, \*, \* }

Nuevo socket: { TCP, IP fuente, Puerto fuente, IP dest., Puerto dest. }



## Características de la llamada ACCEPT.

- Si el servidor es :
  - Iterativo**, tras atender una conexión de cliente, la cierra y vuelve a esperar una nueva conexión sobre el *socket* maestro.
  - Concurrente**, creará un nuevo proceso por cada conexión que se establezca (pasándole el nuevo *socket*), mientras que utiliza el *socket* maestro para recibir nuevas peticiones de conexión.



## Ejemplo de uso de ACCEPT

```
#define port 7 /* Puerto bien-conocido del servicio de ECHO */

int sd, newsd, err, addrlen;
struct sockaddr_in servaddr, cliaddr;

sd = socket (AF_INET, SOCK_STREAM, 0);
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(port);
err = bind (sd, (struct sockaddr *) &servaddr, sizeof(servaddr));
err = listen (sd, 5);
for (;;) {
    newsd = accept (sd, (struct sockaddr *) &cliaddr, &addrlen);
    if (newsd < 0) err_exit ("Error estableciendo la conexión");
    if (fork () == 0) { (* Nuevo proceso *)
        close (sd);
        echo (newsd); (* Función que realiza el servicio de ECHO *)
        exit (0);
    }
    close (newsd); (* Proceso padre *)
}
```



## Llamadas de transferencia de datos

- Existen un conjunto de llamadas que se usan para el envío y recepción de datos a través de un *socket*:
  - Recepción: **read**, **recv**, y **recvfrom**
  - Transmisión: **write**, **send** y **sendto**
- Devuelven el número de octetos que realmente se han leído o escrito.
- Devuelven un “-1” cuando se produce un error.
- Las lecturas **read** y **recv** devuelven un cero cuando se encuentra con la marca fin de fichero (cierre conexión remota).
- Las lecturas eliminan la información del buffer interno del *socket*, pasándola al buffer que suministra la aplicación en la llamada.
- Son bloqueantes: no devuelven el control hasta que no se ha completado la transmisión ó han llenado el buffer de recepción.
  - Es posible modificar este comportamiento en recepción con el bit PUSH del segmento TCP.



## read y write

- Características de **read** y **write**
  - Son las llamadas al sistema clásicas.
  - Se utilizan sobre *sockets* “conectados”.
    - No es necesario indicar la dirección del otro extremo

```
#include <sys/types.h>
#include <unistd.h>
```

```
int read (int sd, char *buff, size_t count);
int write (int sd, char *buff, size_t count);
```

**sd**: Descriptor de socket sobre el que vamos a leer/escribir.  
**buff**: Buffer donde están los datos a enviar/recibir.  
**count**: Número de octetos que queremos leer/escribir.



## send y recv

- Características muy similares a las anteriores.
  - Se usan para realizar envíos o lecturas de datos con condiciones especiales.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int send (int sd, void *buff, int count, int flags);
int recv (int sd, void *buff, int count, int flags);
```

**sd, buff y count**: Mismo significado que Read y Write.

**flags**: Indican un envío o recepción especial. Así:

0 : Envío normal (equivalente a Read/Write).

MSG\_OOB: Envío de datos urgentes.

MSG\_PEEK: Para recoger datos sin eliminarlos del buffer



## sendto y recvfrom

- Se utilizan sobre *sockets* “no conectados” (UDP).
  - Es necesario especificar la dirección del otro extremo.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int recvfrom (int sd, void *buff, int len, int flags, struct
               sockaddr *from, int *fromlen);
int sendto (int sd, void *buff, int len, int flags, struct
             sockaddr *to, int tolen);
```

**sd, buff, count y flags**: Mismo significado que las anteriores.

**from/to** : Direcciones de socket del otro extremo.

**fromlen/tolen**: Tamaño en bytes de la dirección de socket.



## Llamada CLOSE

- Elimina un *socket*.
  - Si el *socket* es de tipo SOCK\_STREAM, se insta a TCP para iniciar el cierre de conexión.
    - Si todavía quedan datos por enviar en los *buffers*, TCP trata de enviarlos antes de indicar el cierre.
  - En cualquier caso, elimina el socket y todos los recursos asociados: *Buffers*, puerto asignado, etc.

```
#include <sys/unistd.h>

int close (int sd);

sd: Descriptor de socket que queremos eliminar.
```



## Llamada SHUTDOWN

- Sólo para *sockets* de tipo SOCK\_STREAM.
- Realiza un cierre de conexión más detallado:
  - En un determinado sentido o en ambos.
  - No elimina el *socket*.

```
#include <sys/socket.h>

int shutdown (int sd, int howto);

sd: Descriptor de socket que queremos eliminar.
howto:
    "0" se cierra el stream de entrada.
    "1" se cierra el stream de salida.
    "2" se cierran los dos streams.
```



## 2.1 Diferencias entre BSDSocket y WinSockets

- Inicialización: *WSAStartup(...)* y *WSACleanup()*
- Tipo *SOCKET*.
- Constantes predefinidas
  - *INVALID\_SOCKET* y *SOCKET\_ERROR*
- Obtención de error: *WSAGetLastError()*
- Algunas primitivas cambian su nombre:
  - *Closesocket()* en vez de *close()*.
  - *Ioctlsocket()* en vez de *ioctl()*.
- Otras no existen:
  - *read* y *write* (en su lugar *send* y *recv*)



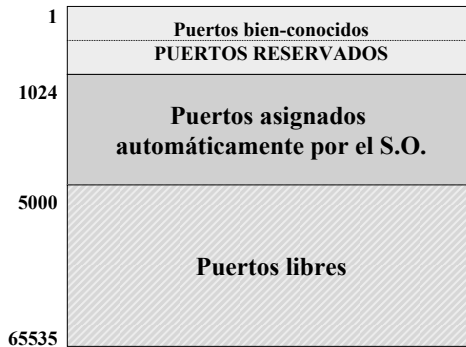
## 3. Asignación de puertos a procesos.

- Las aplicaciones tienen dos alternativas:
  - Pedirle al sistema que asigne un puerto determinado al *socket* que va a utilizar (asignación estática).
    - La usan los servidores (puertos bien-conocidos).
  - Relegar la elección del puerto al sistema operativo (asignación dinámica).
    - En general, la usan los clientes (*BIND*, *port* = 0)
    - El sistema operativo dispone de un espacio de direcciones de puerto que asigna dinámicamente a los procesos que utilizan esta alternativa.



### 3.1 Espacio de direcciones de puerto

- Se disponen de 64K direcciones de puerto
- Los sistemas UNIX dividen el espacio de direcciones de puerto:



### 4. Atributos de los sockets

- Son un conjunto de parámetros asociados a un socket que definen su comportamiento.
- Los atributos se agrupan en clases.
- Las aplicaciones pueden consultar o modificar los atributos de un socket a través de las llamadas:
  - **Getsockopt()** y **Setsockopt()**.

```
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(int sd, int class, int optname,
void *optval, int *optlen);
int setsockopt(int sd, int class, int optname,
void *optval, int optlen);
```



### Atributos de los sockets

Clase	Nombre	Descripción	Get	Set	Bool
IPPROTO_IP	IP_OPTIONS	Opciones de la cabecera IP	Si	Si	No
	TCP_MAXSEG	Tamaño máximo del segmento TCP	Si	No	No
IPPROTO_TCP	TCP_NODELAY	Envíos de segmentos sin retraso.	Si	Si	Si
	SO_BROADCAST	Permite el envío de difusiones	Si	Si	Si
SOL_SOCKET	SO_DEBUG	Activa la recogida de datos estadísticos	Si	Si	Si
	SO_ERROR	Entrega el último código de error.	Si	No	No
	SO_KEEPALIVE	Mecanismo que mantiene la conexión.	Si	Si	Si
	SO_LINGER	Alternativa en el cierre de conexión.	Si	Si	No
	SO_RCVBUF	Tamaño del buffer de recepción.	Si	Si	No
	SO_SNDBUF	Tamaño del buffer de transmisión.	Si	Si	No
	SO_TYPE	Indica el tipo de socket.	Si	No	No
	SO_REUSEADDR	Permite reutilizar direcciones de puerto.	Si	Si	Si
	SO_OOBINLINE	Deja los datos urgentes en el buffer principal.	Si	Si	Si