Sockets (TCP)

Tema 2.- Nivel de aplicación en Internet

Dr. Daniel Morató
Redes de Computadores
Ingeniero Técnico en Informática de Gestión, 2º curso

Material parcialmente adaptado del libro *Computer Networking: A Top Down Approach Featuring the Internet*, 3rd edition. Jim Kurose, Keith Ross, Ed. Addison-Wesley, Julio 2004

Temario

0.- Presentación de la asignatura
1.- Introducción
2.- Nivel de aplicación en Internet
3.- Nivel de transporte en Internet
4.- Nivel de red en Internet
5.- Nivel de enlace

Temario

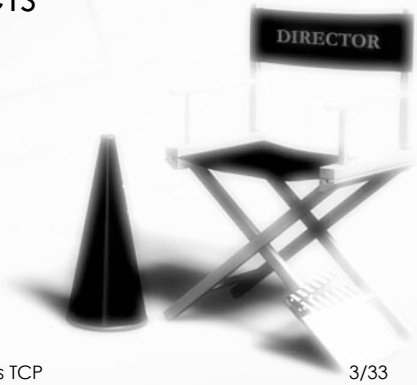
0.- Presentación de la asignatura
1.- Introducción
2.- Nivel de aplicación en Internet
- Principios
- Funcionamiento de servicios
- Diseño y programación de servicios

3.- Nivel de transporte en Internet
4.- Nivel de red en Internet
5.- Nivel de enlace

Tema 2: Servicios

Objetivo:

» Aprender a construir aplicaciones cliente/servidor que se comunican empleando sockets



Programación con Sockets

API de Sockets

» Introducida en el UNIX BSD4.2 en 1983

» Centrada en el paradigma cliente/servidor

» Ofrece dos tipos de servicios de transporte:
- STREAM: flujo de datos fiable orientado a conexión
- DGRAM: datagramas

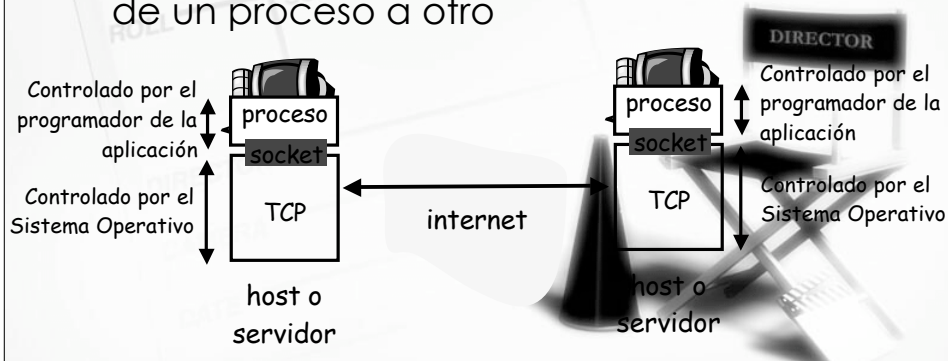
socket

Un interfaz local al host, creado por la aplicación, controlado por el S.O., mediante el cual procesos de aplicación pueden enviar y recibir mensajes hacia/desde otros procesos de aplicación

Programación con Sockets TCP

Socket: una puerta entre el proceso de aplicación y el protocolo de transporte (UCP o TCP)

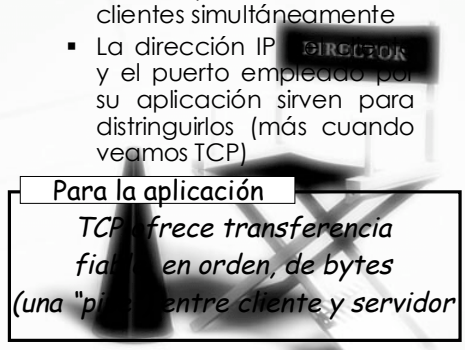
Servicio TCP: transferencia fiable de **bytes** de un proceso a otro



Programación con Sockets TCP

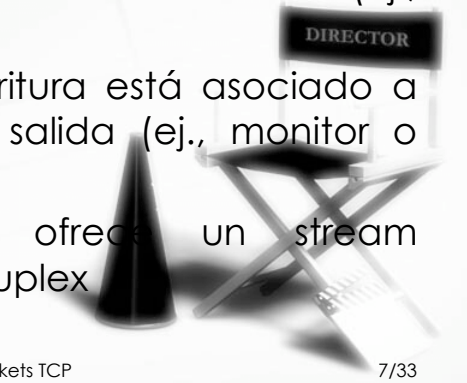
El cliente debe contactar con el servidor
- » El servidor debe estar ejecutándose primero
- » El servidor debe haber creado un socket por el que espera que el cliente contacte con él
- » El cliente crea su propio socket
- » Especifica la dirección IP del servidor y el puerto de la aplicación
- » Entonces se establece la conexión TCP con el servidor

» Cuando el servidor es contactado crea un nuevo socket TCP para la comunicación con el cliente
- Permite que el servidor se comunique con varios clientes simultáneamente
- La dirección IP y el puerto empleado por su aplicación sirven para distringuirlos (más cuando veamos TCP)



Stream

- » Un stream es una secuencia de bytes que fluye hacia o desde un proceso
- » Un stream de lectura se encuentra asociado a un dispositivo de entrada (ej., teclado, socket)
- » Un stream de escritura está asociado a un dispositivo de salida (ej., monitor o socket)
- » Un socket TCP ofrece un stream bidireccional full duplex



Sockets TCP

» Cliente TCP
» Servidor TCP

Creación de un Socket

```
int socket(int domain, int type, int protocol)
```

» `int domain`
  - Hay diferentes tipos de sockets para diferentes familias de protocolos

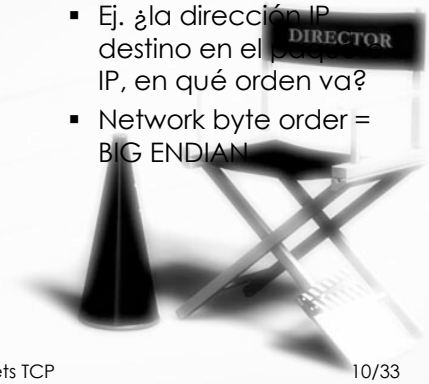
» `int type`
  - `SOCK_STREAM`, `SOCK_DGRAM`, (otros)

» `int protocol`
  - En caso de que haya varios protocolos en la misma categoría

Network Order vs Host Order

» Host byte order
  - Los números de más de un byte se pueden guardar en memoria empezando
    - » Por el más significativo: BIG ENDIAN
    - » Por el menos significativo: LITTLE ENDIAN
  - Las CPUs normalmente soportan solo un modo o al menos solo emplean uno
    - » Intel, AMD: LITTLE ENDIAN
    - » SPARC, Motorola, IBM: BIG ENDIAN

» Network byte order
  - Máquinas de diferentes arquitecturas deben "entenderse"
  - Ej. ¿la dirección IP destino en el paquete IP, en qué orden va?
  - Network byte order = BIG ENDIAN



Network Order vs Host Order

Network Order -> Host order

» `int ntohl(int)`
  - Para 32bits

» `short ntohs(short)`
  - Para 16bits

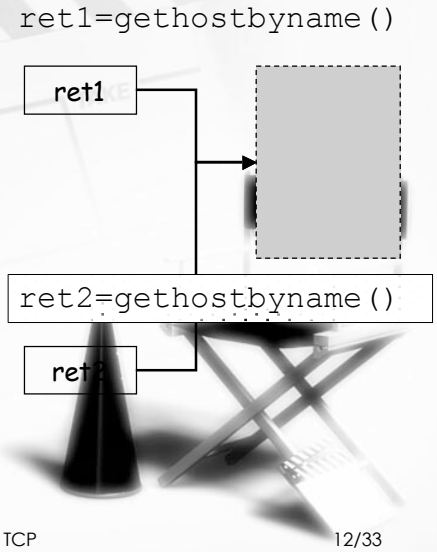
Host Order -> Network order

» `int htonl(int)`
  - Para 32bits

» `short htons(short)`
  - Para 16bits

```
struct hostent *gethostbyname(char *)
```

- DNS
- Se le pasa una cadena con un nombre de dominio
  - "www.tlm.unavarra.es"
- o una cadena con el nombre en formato "dot-decimal"
  - "130.206.160.215"
- Devuelve un puntero a una estructura
- Dicha estructura existe dentro de la librería
- Siempre es la misma
- Entre sus campos está la IP (en network byte order)



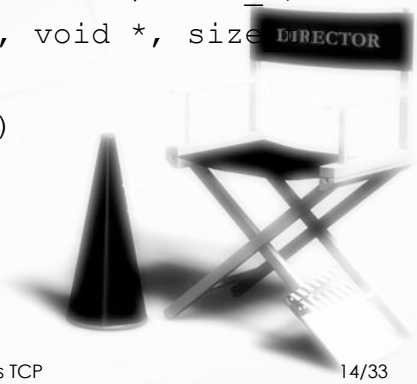
Conectar

```
int connect(int s, struct sockaddr *name, int len)
```

- `int s`
  - Socket
- `struct sockaddr *name`
  - Identifica al socket del otro con su:
    - Dirección IP
    - Puerto
- `int len`
  - Tamaño (bytes) de la estructura a la que apunta el segundo argumento

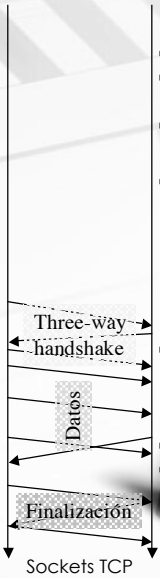
El Socket

- Es un *descriptor de fichero* (file descriptor)
- Se pueden emplear con él funciones típicas que trabajan con descriptores
  - `ssize_t read(int, void *, size_t)`
  - `ssize_t write(int, void *, size`
  - `int dup(int)`
  - `int dup2(int, int)`
  - `int close(int)`
  - `select()`

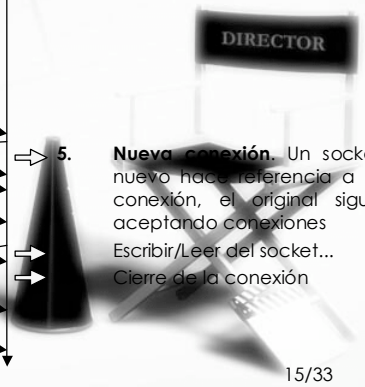


Ejemplo en pseudo-código

- Cliente
  1. Crear el socket TCP (Stream)
  2. Solicitar al S.O. que lo conecte con un destino (IP+puerto) concreto
  3. **Conexión establecida**
  4. Escribir/Leer del socket...
  5. Cerrar el socket/conexión



- Servidor
  1. Crear el socket TCP (Stream)
  2. Asignarle el puerto en el que esperar
  3. Solicitar al S.O. que escuche y acepte esas conexiones
  4. Esperar una conexión...
  5. **Nueva conexión.** Un socket nuevo hace referencia a la conexión, el original sigue aceptando conexiones
  6. Escribir/Leer del socket...
  7. Cierre de la conexión



Cliente en C (1)

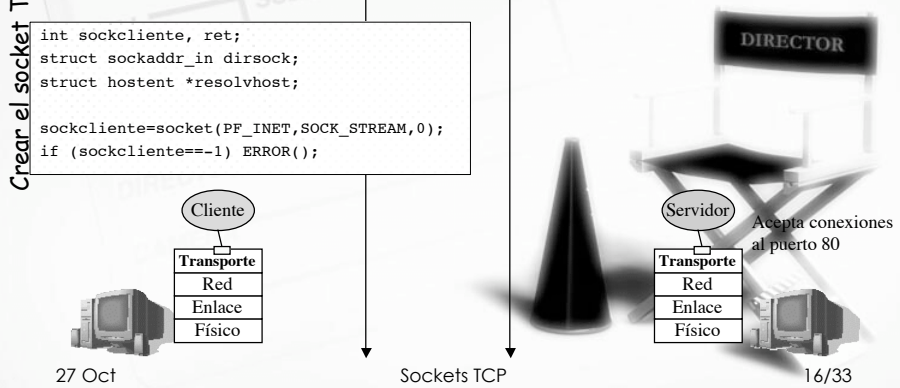
» Cliente

» Servidor

1. Crear el socket TCP (Stream)
2. Asignarle el puerto en el que esperar
3. Solicitar al S.O. que escuche y acepte esas conexiones
4. Esperar una conexión...

Crear el socket TCP

```
int sockcliente, ret;
struct sockaddr_in dirsock;
struct hostent *resolvhost;

sockcliente=socket(PF_INET,SOCK_STREAM,0);
if (sockcliente==-1) ERROR();
```



Cliente en C (2)

» Cliente

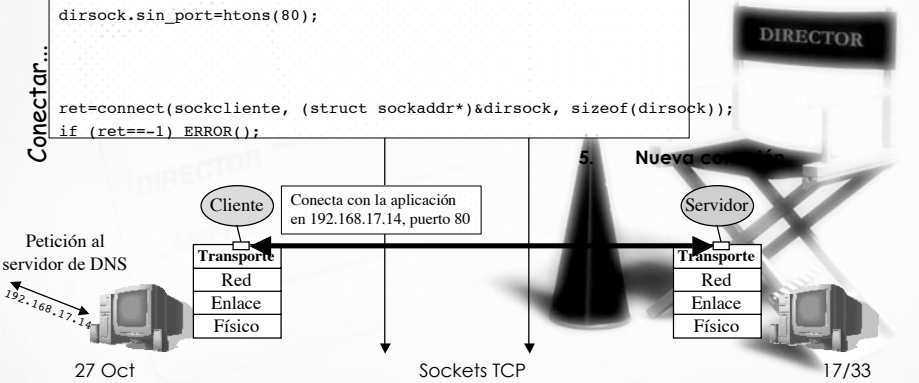
» Servidor

4. Esperar una conexión

DNS...

Conectar...

```
dirsock.sin_family=AF_INET;
resolvhost=gethostbyname("servidor.tlm.unavarra.es");
if (resolvhost==NULL) ERROR();
dirsock.sin_addr.s_addr=*(u_long*)resolvhost->h_addr_list[0];

dirsock.sin_port=htons(80);



ret=connect(sockcliente, (struct sockaddr*)&dirsock, sizeof(dirsock));
if (ret==-1) ERROR();
```



Cliente en C (y 3)

» Cliente

Enviar/recibir

```
write(sockcliente,...);
read(sockcliente,...);
.
.
.
.
.
.
.
```

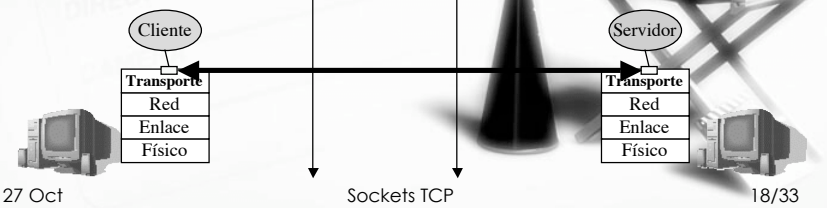
Cierre

```
close(sockcliente);
```

» Servidor

6. Escribir/Leer del socket...

7. Cierre de la conexión...



Compilación

» ¿Headers?

- `#include <sys/types.h>`
- `#include <sys/socket.h>`
- `#include <netdb.h>` *(gethostbyname())*
- `#include <netinet/in.h>` *(htons())*

» ¿Librerías?

- Según el UNIX: ninguna, `-lsocket`, `-lnsl` ...

Pruébelo

» Cree un cliente que se conecte a un servidor Web

» Que le solicite una página Web mandando un mensaje de petición de HTTP

Sockets TCP

» Cliente TCP

» Servidor TCP

Socket

» Creación con
  - `int socket(int, int, int)`
  - Igual que en el extremo del cliente

"Asociar" a un puerto

`int bind(int s, struct sockaddr *name, int len)`

» Da "nombre" a un socket

» `int s`
  - El socket

» `struct sockaddr *name`
  - El puerto en el que "escucha" el servidor
  - Si el host tuviera varias direcciones IP (varios NIC) también se podría seleccionar una

» `int len`
  - Tamaño de la estructura a la que apunta el segundo argumento

Empezar a "escuchar"

`int listen(int s, int backlog)`

» Indica al S.O. que empiece a aceptar conexiones al puerto al que está asociado el socket

» `int s`
- El socket

» `int backlog`
- El S.O. acepta conexiones sin interacción de la aplicación servidor
- Limita cuántas puede tener aceptadas sin que el programa las atienda

Atender a la conexión

`int accept(int s, struct sockaddr *addr, int *len)`

» Devuelve un nuevo socket

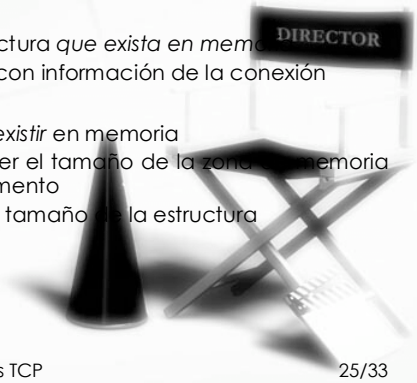
» Está asociado a una de las conexiones aceptadas por el S.O.

» Una de las conexiones en la cola del *listen()*

» `int s`
- El socket

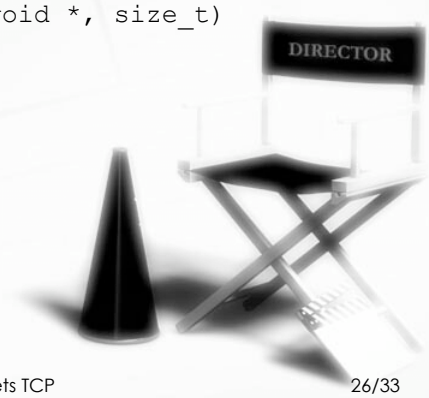
» `struct sockaddr *addr`
- Debe ser el puntero a una estructura *que exista en memoria*
- La función rellena la estructura con información de la conexión

» `int *len`
- Puntero a un entero *que debe existir* en memoria
- Al llamar a la función debe tener el tamaño de la zona de memoria apuntada por el segundo argumento
- Al salir de la función contiene el tamaño de la estructura



El Socket

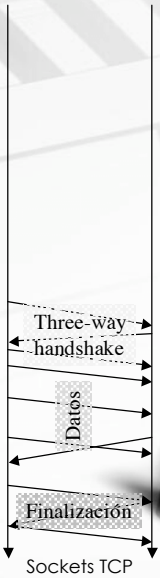
- Es un *descriptor de fichero* (file descriptor)
- Se pueden emplear con él funciones típicas que trabajan con descriptores
  - `ssize_t read(int, void *, size_t)`
  - `ssize_t write(int, void *, size_t)`
  - `int dup(int)`
  - `int dup2(int, int)`
  - `int close(int)`
  - `select()`



Ejemplo en pseudo-código

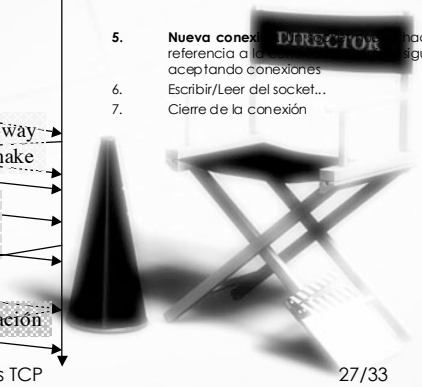
- Cliente

1. Crear el socket TCP (Stream)
2. Solicitar al S.O. que lo conecte con un destino (IP+puerto) concreto
3. **Conexión establecida**
4. Escribir/Leer del socket...
5. Cerrar el socket/conexión



- Servidor

1. Crear el socket TCP (Stream)
2. Asignarle el puerto en el que esperar
3. Solicitar al S.O. que escuche y acepte esas conexiones
4. Esperar una conexión...
5. **Nueva conexi** hace referencia a la sigue aceptando conexiones
6. Escribir/Leer del socket...
7. Cierre de la conexión



Servidor en C (1)

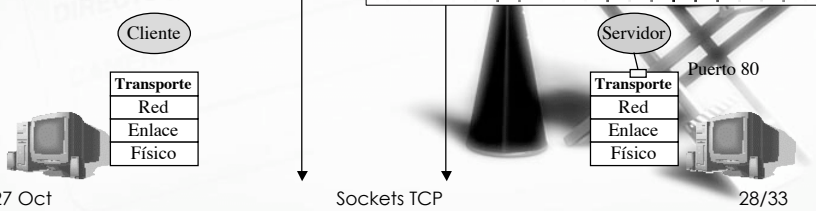
» Cliente

» Servidor

```
int sockservidor, sockconectado;
int ret, dirlen=sizeof(dirsock);
struct sockaddr_in dirsock;

sockservidor=socket(PF_INET,SOCK_STREAM,0);
if (sockservidor==-1) ERROR();


dirsock.sin_family=AF_INET;
dirsock.sin_addr.s_addr=INADDR_ANY;
dirsock.sin_port=htons(80);

ret= bind(sockservidor, (struct sockaddr*)&dirsock,
     sizeof(dirsock));
if (ret==-1) ERROR();
```

Crear el socket TCP...

"bind"...



Servidor en C (2)

» Cliente

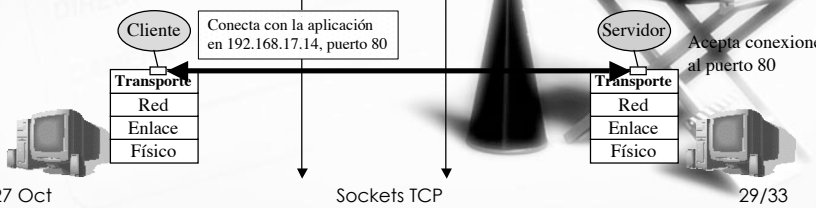
» Servidor

```
ret=listen(sockservidor,5);
if (ret==-1) ERROR();


sockconectado=accept(sockservidor, (struct
      sockaddr*)&dirsock, &dirlen);
```

"listen"...

Aceptar conexión...

1. Crear el socket TCP (Stream) ...
2. Solicitar al S.O. que lo conecte con un destino (IP+puerto) concreto ...



Servidor en C (y 3)

» Cliente

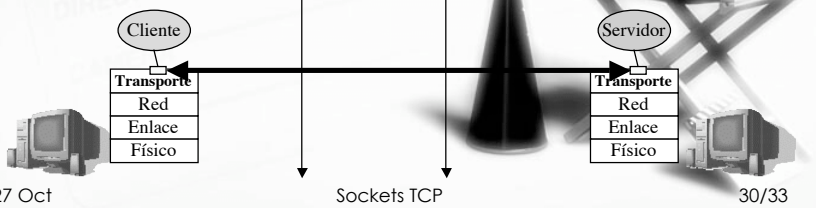
1. Escribir/Leer del socket
2. Cerrar el socket/conexión

» Servidor

```
write(sockconectado,...);
read(sockconectado,...);
.
.
.
.
.
.
.
.
.
close(sockconectado,...);
```

Enviar/Recibir

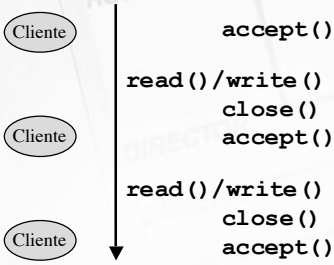
Cerrar conexión...



Tipos de servidores

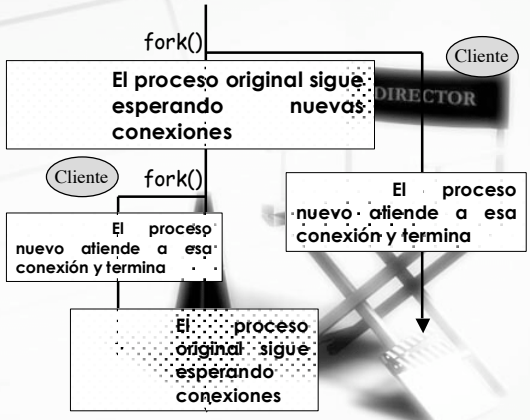
» Servidor iterativo

1. Acepta una conexión
2. Realiza la comunicación
3. Cierra la conexión
4. Vuelve al punto 1.



» Servidor concurrente

- Acepta una conexión
- Crea un nuevo *proceso*



Servidores concurrentes

» Permiten atender varias peticiones simultáneamente
» No se detiene porque un servicio sea muy "largo"
  - Gran fichero a servir
  - Transferencia lenta

» Concurrencia:
  - Mediante atención alternada: `select()`
  - Mediante multiproceso
    » `fork()`
    » Se heredan los descriptores de fichero
    » Aprovechan NxCPUs
  - Mediante multihilo
    » threads
    » Comparten el espacio de memoria
    » Aprovechan NxCPUs

Próximo día

Sockets UDP