



**ARQUITECTURA DE REDES, SISTEMAS Y SERVICIOS**  
*Área de Ingeniería Telemática*

# Desarrollo de clientes y servidores

Area de Ingeniería Telemática  
<http://www.tlm.unavarra.es>

Arquitectura de Redes, Sistemas y Servicios  
3º Ingeniería de Telecomunicación



# Temario

1. Introducción
2. Protocolos y arquitectura
3. Redes de área local
4. Protocolos de Internet
5. Conmutación de circuitos
6. Conmutación de paquetes
7. Gestión de recursos en conmutadores
8. Protocolos de control de acceso al medio



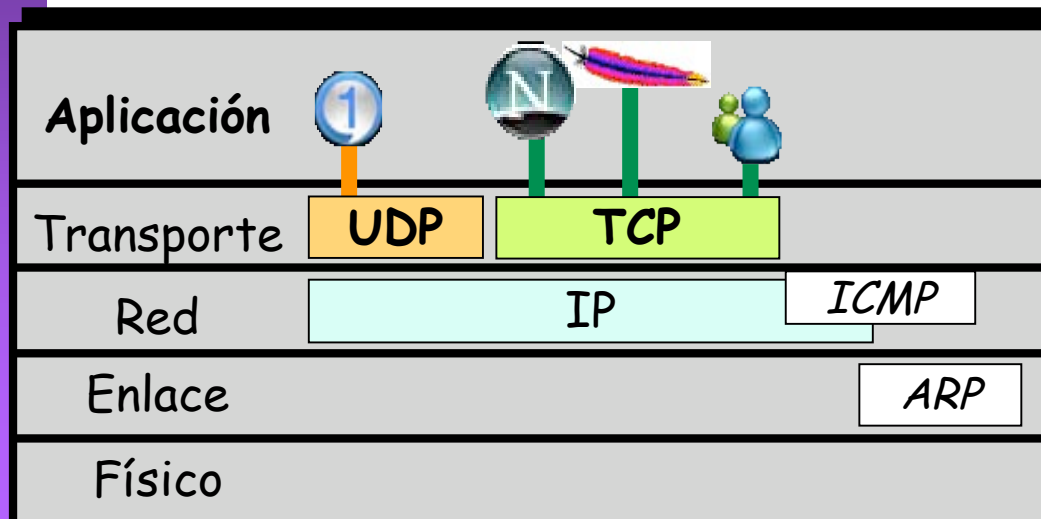
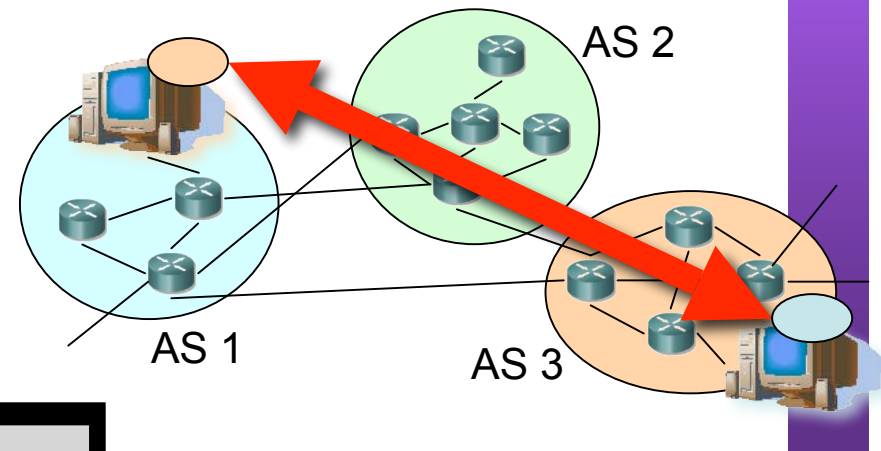
# Temario

1. Introducción
2. Protocolos y arquitectura
3. Redes de área local
4. Protocolos de Internet
  - Nivel de red
  - Nivel de transporte
  - Servicios
    - La Web
    - E-Mail. FTP. Telnet
    - Otros
    - **Desarrollo de clientes y servidores**
5. Conmutación de circuitos
6. Conmutación de paquetes
7. Gestión de recursos en conmutadores
8. Protocolos de control de acceso al medio



# Las aplicaciones

- Son software
- Diferentes máquinas y Sistemas Operativos
- Quienes se comunican son **procesos**

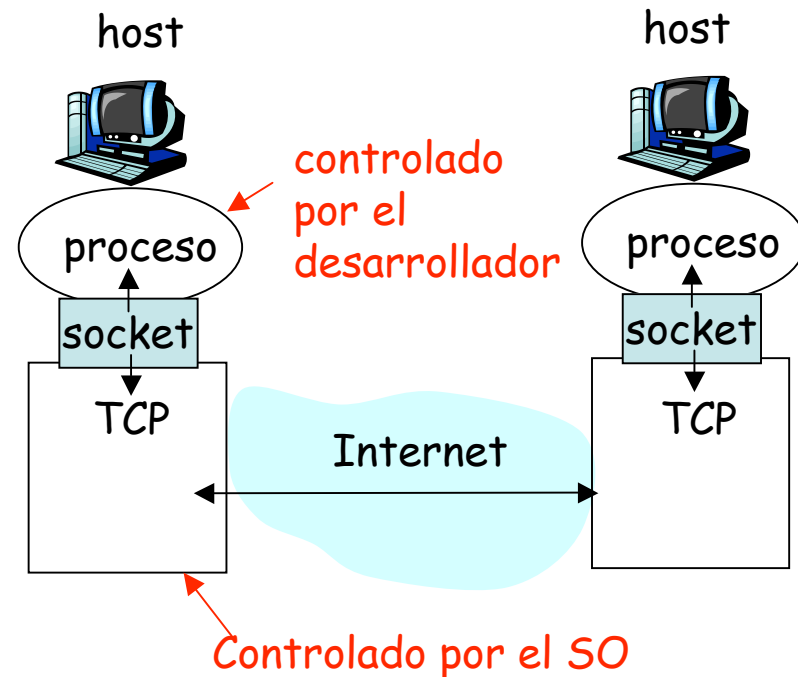


- Intercambian mensajes
- Emplean **Protocolos de nivel de aplicación**



# Sockets

- Los procesos envían y reciben mensajes a través de un **socket**
- Delega en el nivel de transporte para que haga llegar los mensajes al otro socket
- Acceso a través de un **API**
- Puede escoger el protocolo de transporte
- Puede configurar algunos parámetros del mismo
- No controla cómo se comporta





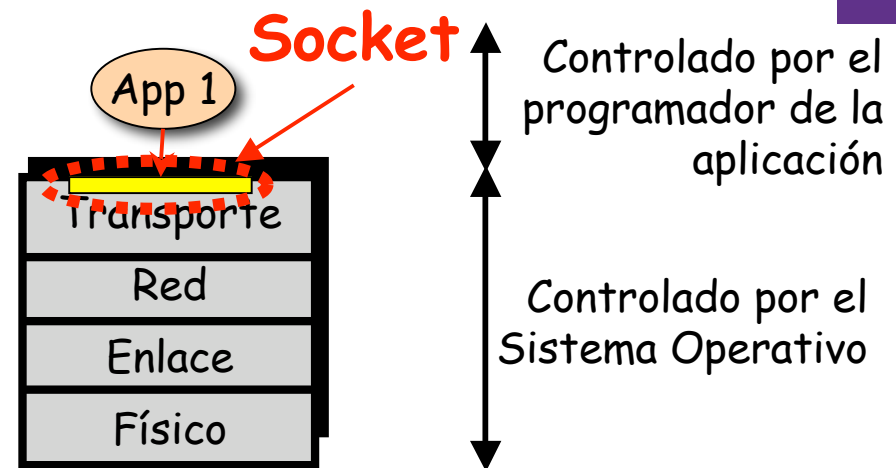
# Programación con Sockets

## 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

- Creado por la aplicación
- Controlado por el S.O.
- A través suya la aplicación envía y recibe mensajes





# Sockets y UDP

## **UDP: no hay “conexión” entre cliente y servidor**

- No hay handshaking
- El emisor debe indicar explícitamente la dirección IP y el puerto del destino para cada paquete
- El servidor debe extraer la dirección IP y el puerto del emisor del paquete

- UDP ofrece transferencia no fiable de grupos de bytes (“datagramas”) entre el cliente y el servidor



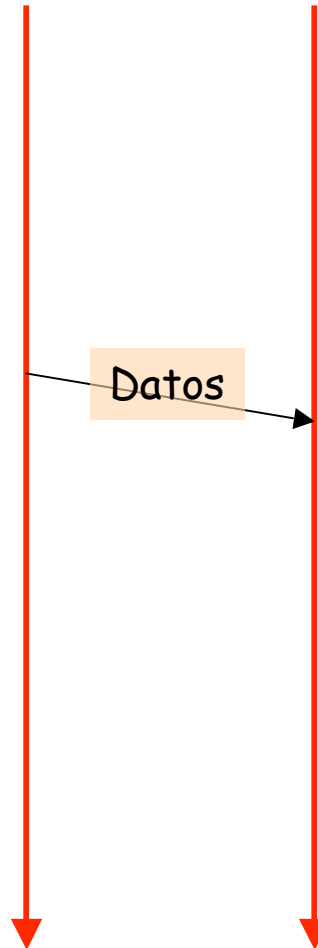
# Ejemplo en pseudo-código

## Cliente

- Crear el socket UDP (Dgram)
- Solicitar al S.O. que se envíen ciertos datos a un destino (IP+puerto) concreto (...)

## Servidor

- Crear el socket UDP (Dgram)
- Asignarle el puerto en el que esperar
- Esperar un datagrama
- Datagrama recibido (o no)







# Ejemplo en C (I)

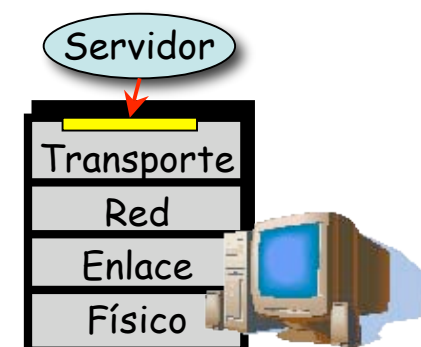
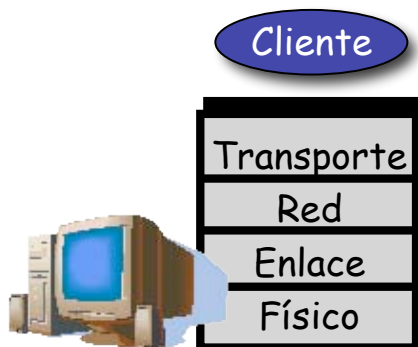
## Cliente

## Servidor

```
struct sockaddr_in dirsock, emisor;  
int sockservidor, ret,  
    frlen=sizeof(emisor);  
char *buf[2000];
```

```
sockservidor=socket(PF_INET, SOCK_DGRAM, 0);  
if (sockservidor==-1) ERROR();  
dirsock.sin_family=AF_INET;
```

Crear el socket UDP (...)





# Ejemplo en C (II)

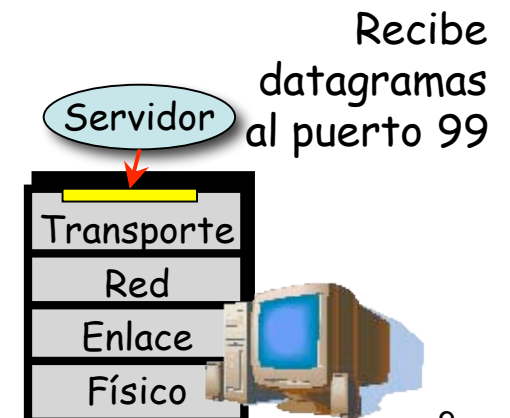
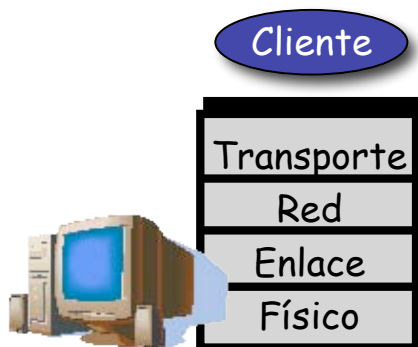
## Cliente

## Servidor

```
dirsock.sin_addr.s_addr=INADDR_ANY;  
dirsock.sin_port=htons(99);  
ret=      bind(sockservidor,      (struct  
          sockaddr*)&dirsock, sizeof(dirsock));  
if (ret==-1) ERROR();  
  
ret=recvfrom(sockservidor, buf, 2000, 0,  
             (struct sockaddr*)&emisor, &frlen);
```

Asignar Puerto (...)

Esperar  
a recibir

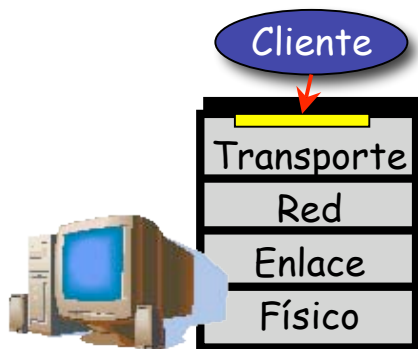




# Ejemplo en C (III)

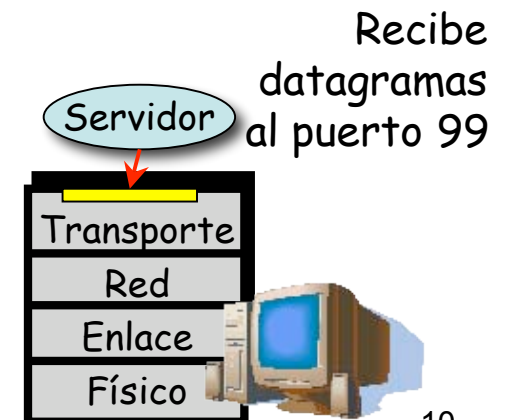
## Cliente

```
int sockcliente, ret;  
struct sockaddr_in dirsock;  
struct hostent *resolvhost;  
  
sockcliente=socket(PF_INET,SOCK_DGRAM,0  
    );  
if (sockcliente==-1) ERROR();
```



## Servidor

- Esperando a recibir





# Ejemplo en C (y IV)

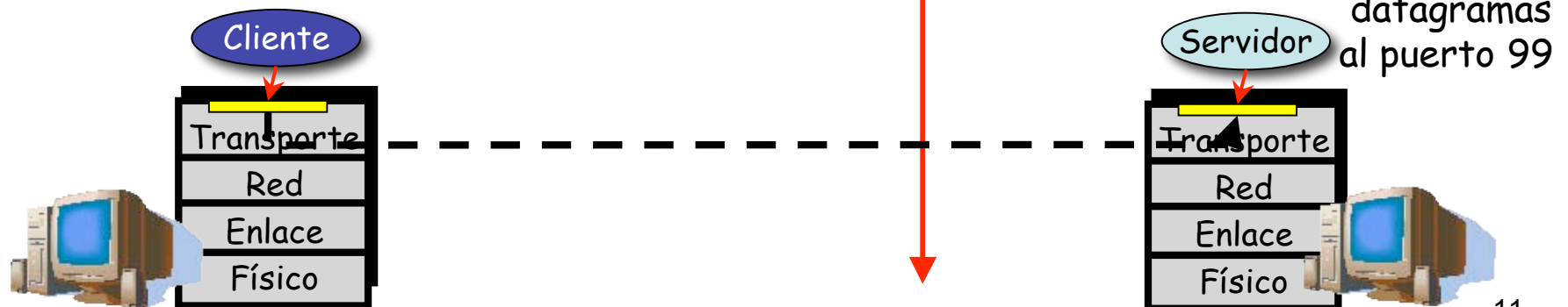
## Cliente

```
dirsock.sin_family=AF_INET;  
resolvhost=gethostbyname("servidor.tlm.unavar  
ra.es");  
if (resolvhost==NULL) ERROR();  
dirsock.sin_addr.s_addr=*(u_long*)resolvhost-  
>h_addr_list[0];  
dirsock.sin_port=htons(99);  
  
ret=sendto(sockcliente, buf, max, 0,  
(struct sockaddr*)dirsock,  
sizeof(dirsock));
```

## Servidor

- Esperando a recibir

```
if (ret==-1) ERROR();
```





# Cliente y Servidor TCP

## El servidor

- Ejecutándose primero
- Debe haber creado un socket por el que espera que el cliente contacte con él
- Indica el puerto asociado

## El cliente

- Crea su propio socket
- Contacta con el servidor
- Especifica la dirección IP del servidor y el puerto de la aplicación
- Se establece la conexión TCP con el servidor

## Servidor al ser contactado

- Crea un nuevo socket TCP para la comunicación con el cliente
- Permite que el servidor se comunique con varios clientes simultáneamente



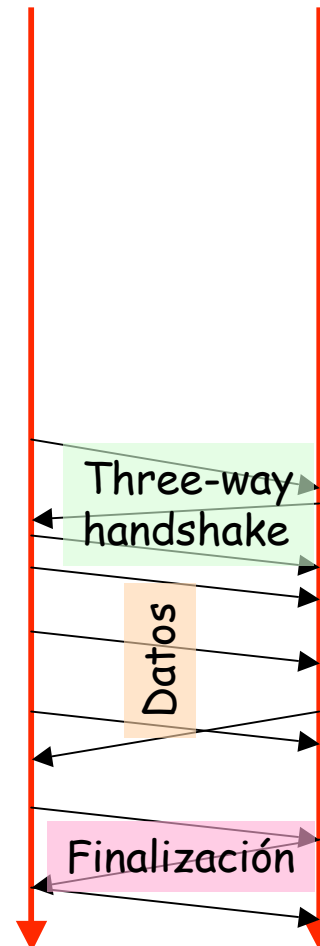
# Ejemplo en pseudo-código

## Cliente

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

## Servidor

- Crear el socket TCP (Stream)
- Asignarle el puerto en el que esperar
- Solicitar al S.O. que escuche y acepte esas conexiones
- Esperar una conexión
- Nueva conexión. Socket nuevo. El original sigue aceptando conexiones
- Escribir/Leer del socket
- Cierre de la conexión

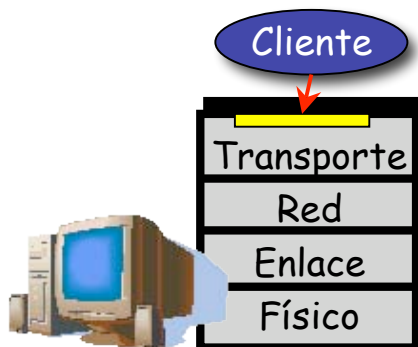




# Cliente en C (I)

## Cliente

```
int sockcliente, ret;  
struct sockaddr_in dirsock;  
struct hostent *resolvhost;  
  
sockcliente=socket(PF_INET, SOCK_STREAM, 0);  
if (sockcliente==-1) ERROR();
```



## Servidor

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





# Cliente en C (II)

## Cliente

```
dirsock.sin_family=AF_INET;  
resolvhost=gethostbyname("servidor.tlm.u  
navarra.es");  
  
if (resolvhost==NULL) ERROR();  
dirsock.sin_addr.s_addr=*(u_long*)resolv  
host->h_addr_list[0];  
  
dirsock.sin_port=htons(80);
```

Petición al  
servidor de DNS

Cliente

192.168.17.14

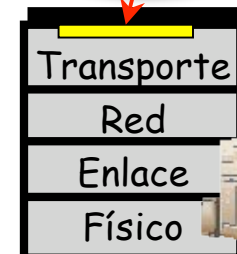


## Servidor

- Esperar una conexión

Acepta conexiones  
al puerto 80

Servidor







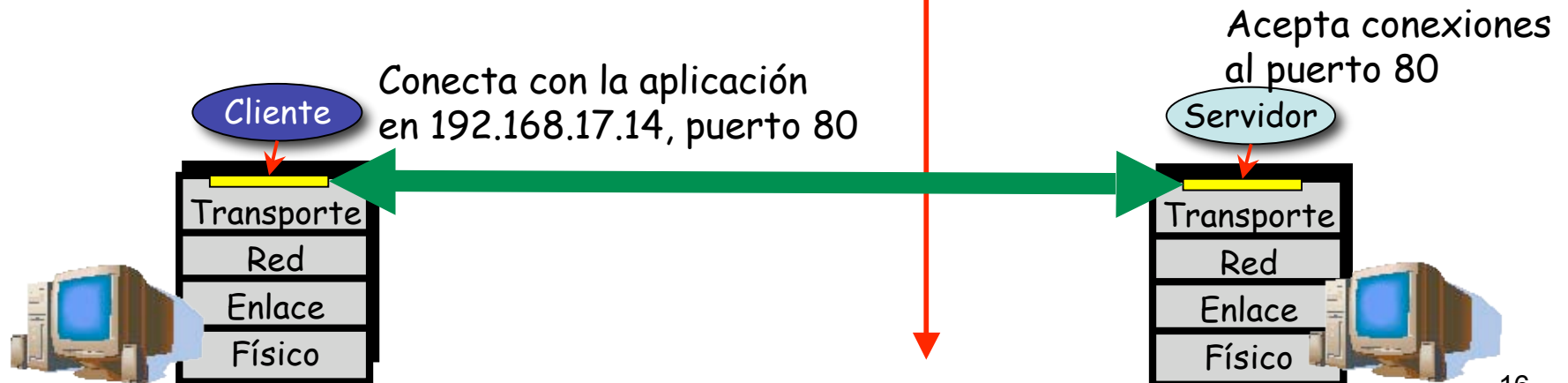
# Cliente en C (III)

## Cliente

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

## Servidor

- Esperar una conexión
- Nueva conexión





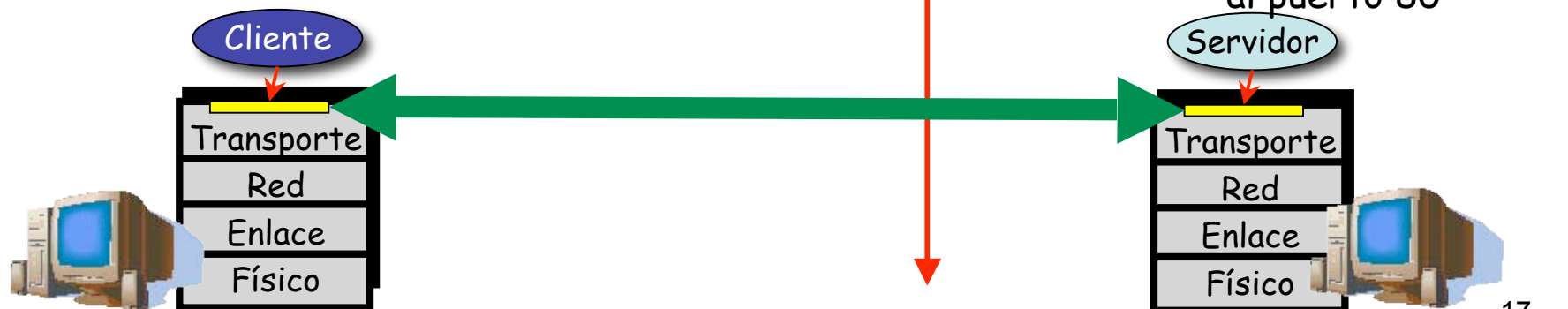
# Cliente en C (y IV)

## Cliente

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

## Servidor

- Escribir/Leer del socket
- Cierre de la conexión





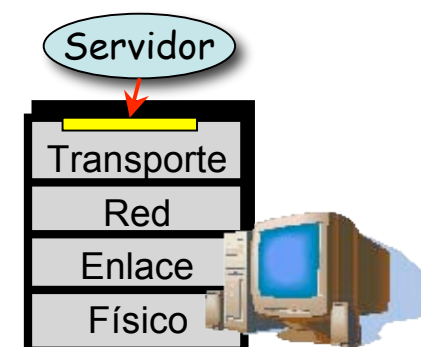
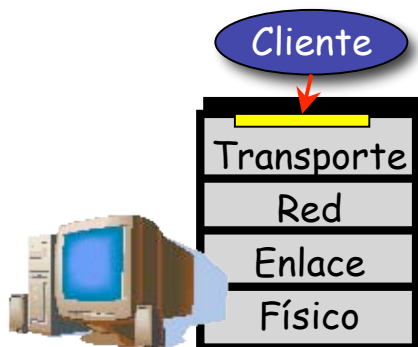
# Servidor en C (I)

## 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();
```

- Crear el socket TCP (Stream)



Crear el socket TCP (...)



# Servidor en C (II)

## Cliente

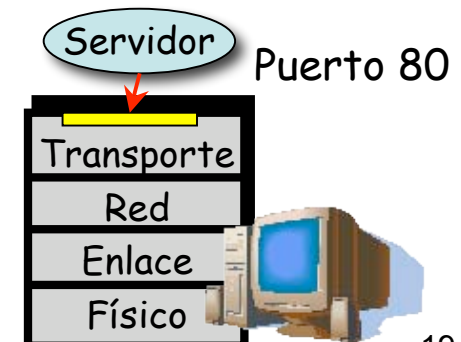
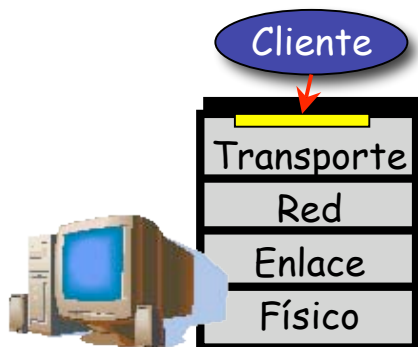
## Servidor

```
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 (Stream)

"bind" (...)





# Servidor en C (III)

## Cliente

- Solicitar al S.O. que lo conecte con un destino (IP+puerto) concreto (...)

## Servidor

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

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

"listen" (...)

Entregar  
conexión





# Servidor en C (y IV)

## Cliente

## Servidor

- Escribir/Leer del socket

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

•  
•  
•  
•

```
close(sockconectado);
```

- Cierre de la conexión

Enviar/Recibir

Cerrar  
Conexión (...)

