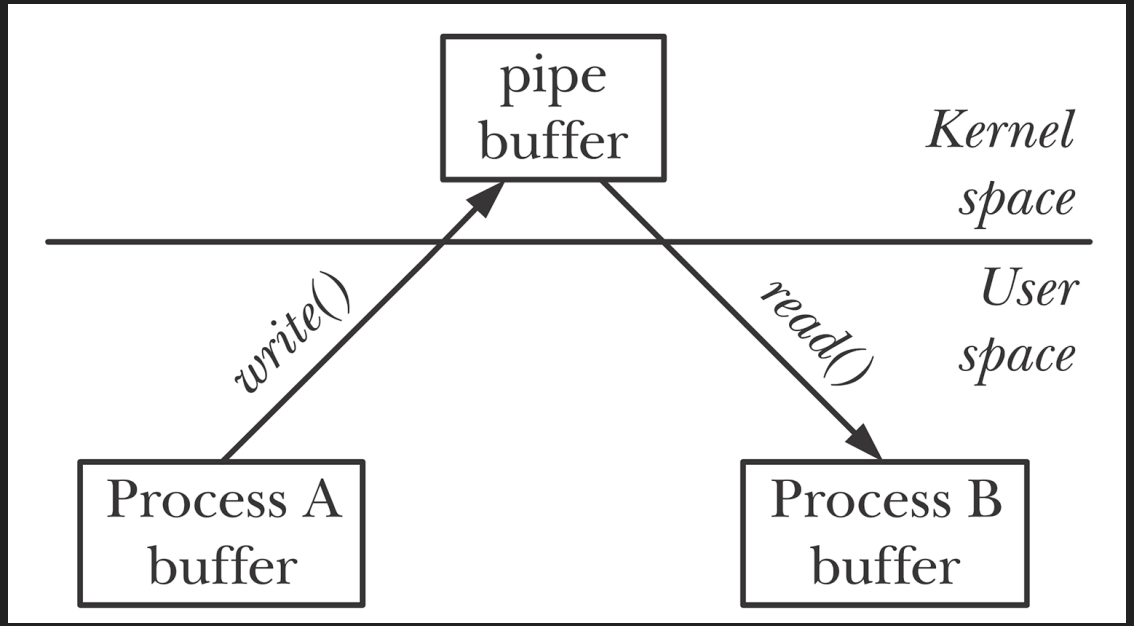
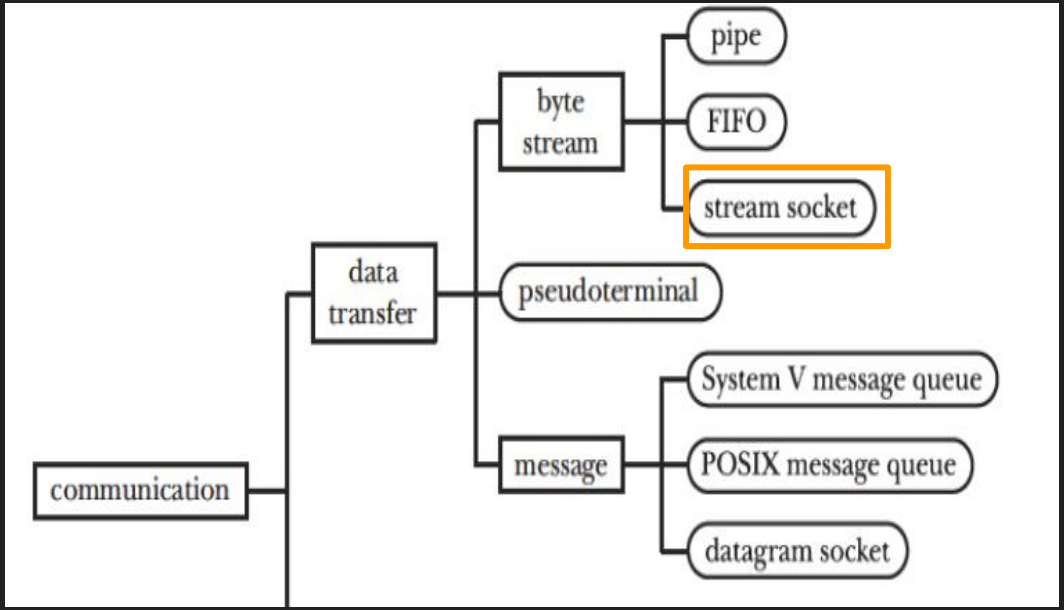
Sockets

¿Cómo podemos comunicar procesos?

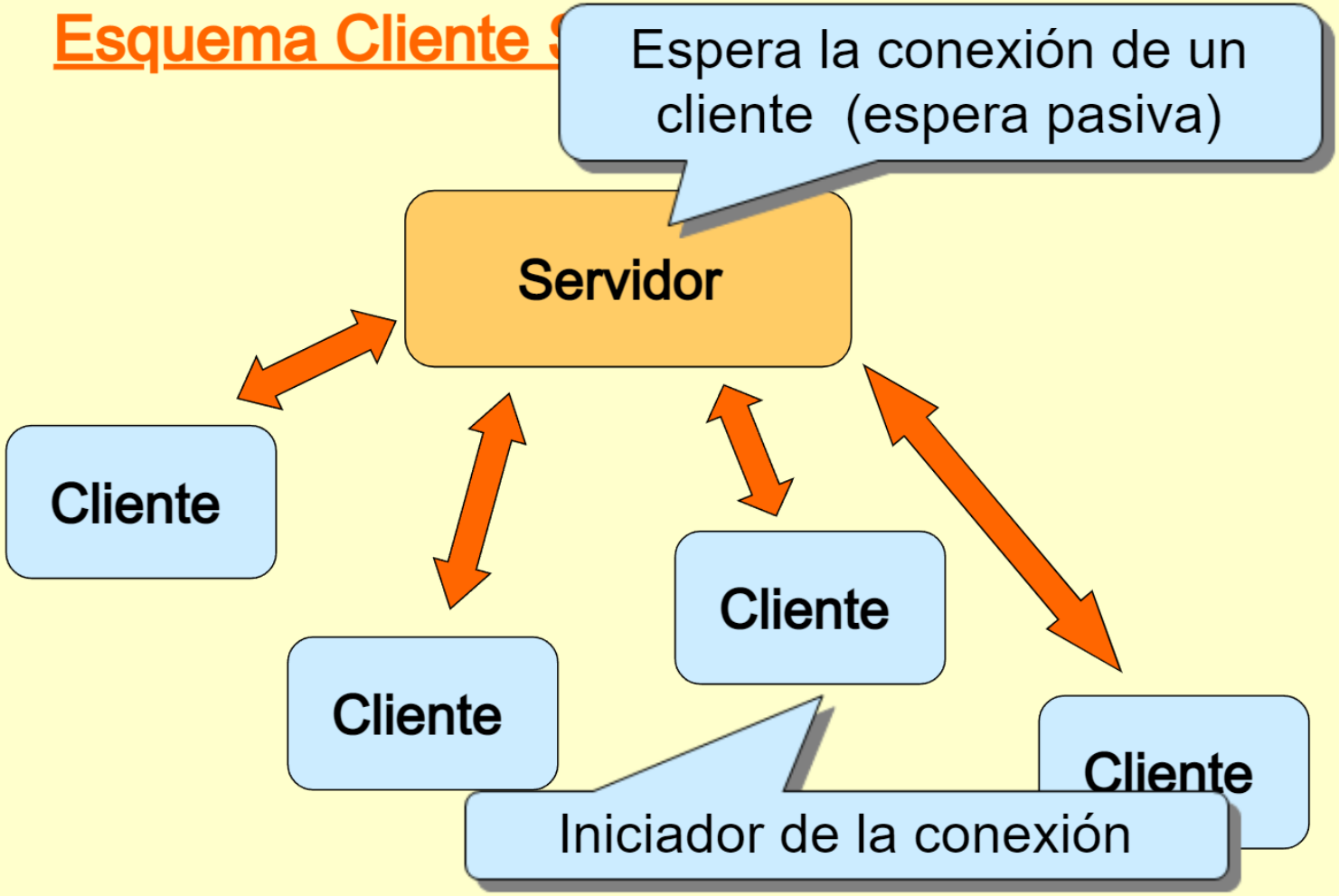
Semántica de la comunicación por transferencia de datos (ejemplo: pipes)

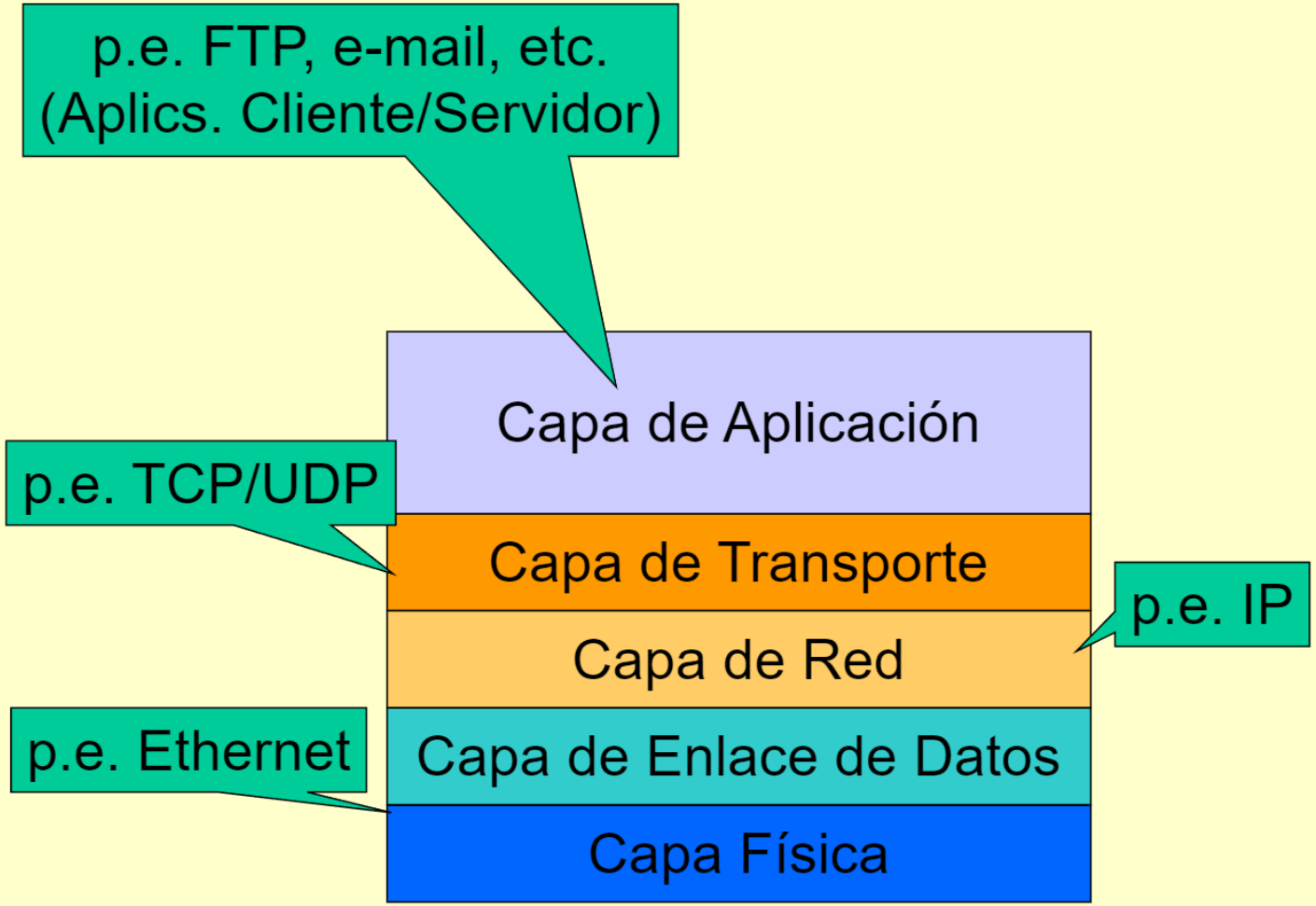


SOCKETS


communication
data transfer
byte stream
pipe
FIFO
stream socket
pseudoterminal
message
System V message queue
POSIX message queue
datagram socket

Esquema Cliente S




p.e. FTP, e-mail, etc.
(Aplics. Cliente/Servidor)
Capa de Aplicación
p.e. TCP/UDP
Capa de Transporte
p.e. IP
Capa de Red
p.e. Ethernet
Capa de Enlace de Datos
Capa Física

Three Kinds of Computer Identifiers

- **Host name** (e.g., www.google.com)
  - Mnemonic name appreciated *by humans*
  - Provides little (if any) information about location
  - Hierarchical, based on organizations
- **IP address** (e.g., 64.236.16.20, fe80::e2:2713:d59c:14ab)
  - Numerical address appreciated *by routers*
  - Hierarchical, based on organizations and topology
- **MAC address** (e.g., 00-15-C5-49-04-A9)
  - Numerical 48-bit address appreciated *by network cards*
  - Non-hierarchical, unrelated to network topology

Try This Now

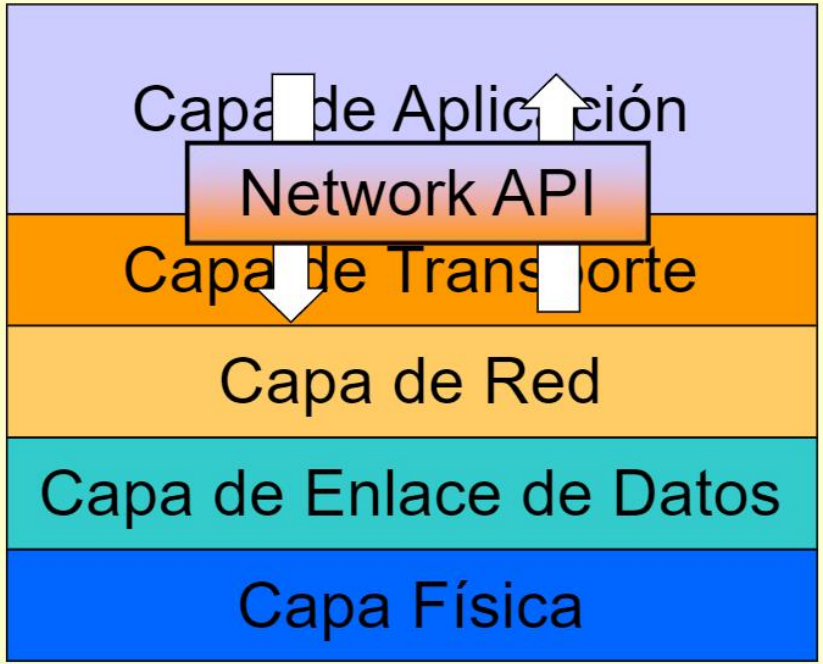
- Unix machine:

```
ifconfig –a
```

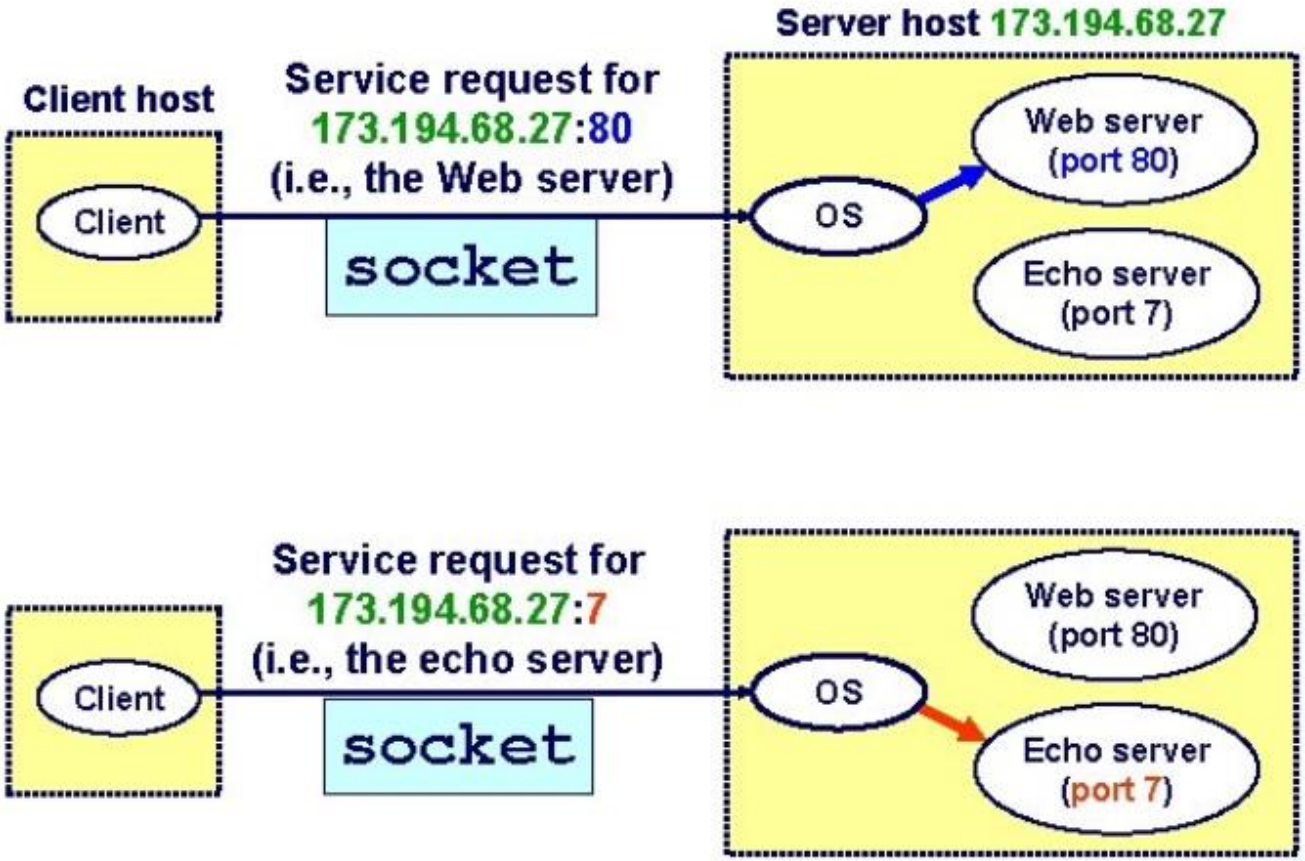
- Windows machine:

```
ipconfig /all
```

La mayoría de los sistemas han adoptado ***Socket API*** (disponible en la mayoría de los S.O., p.e. Linux, UNIX, Windows)

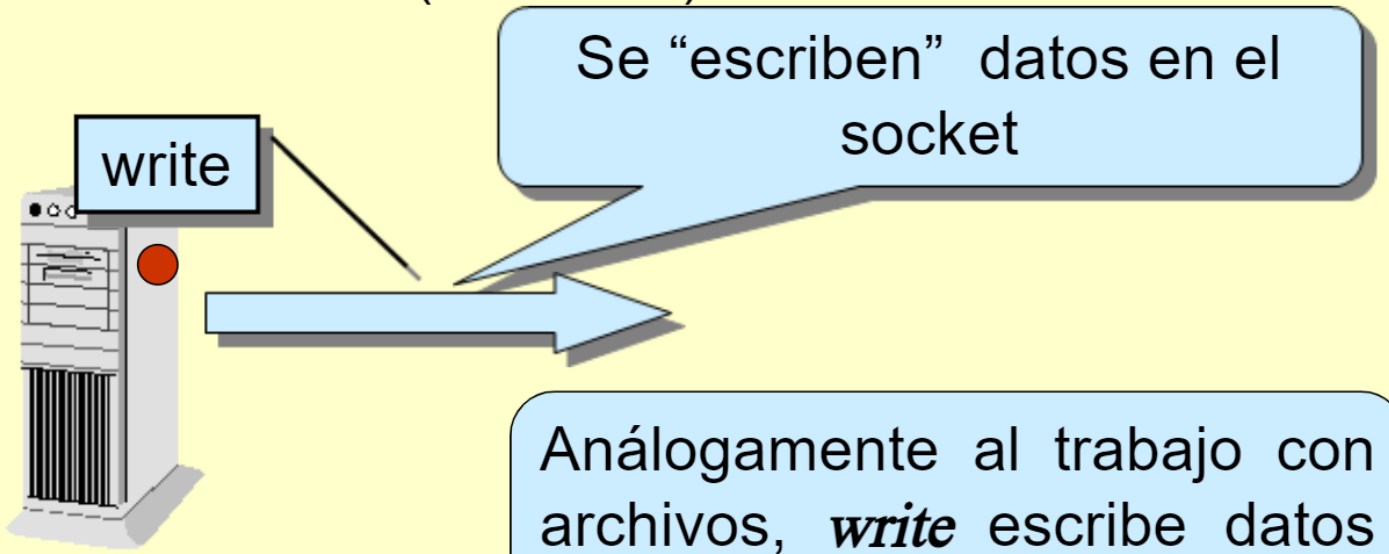


Using Ports to Identify Services



Socket API (sockets)

↖ Los sockets operan según el paradigma *open-read-write-close* (UNIX I/O).



Stream socket

Stream: secuencia de bytes transmitidos en una comunicación orientada a la conexión (TCP)

TCP

PROCESO QUE ESCUCHA (LISTENING PROCESS):

SERVIDOR

PROCESO QUE SE CONECTA (CONNECTOR PROCESS):

CLIENTE

¿Y qué es un socket?

Es una **abstracción** que permite **identificar** un mecanismo para enviar datos

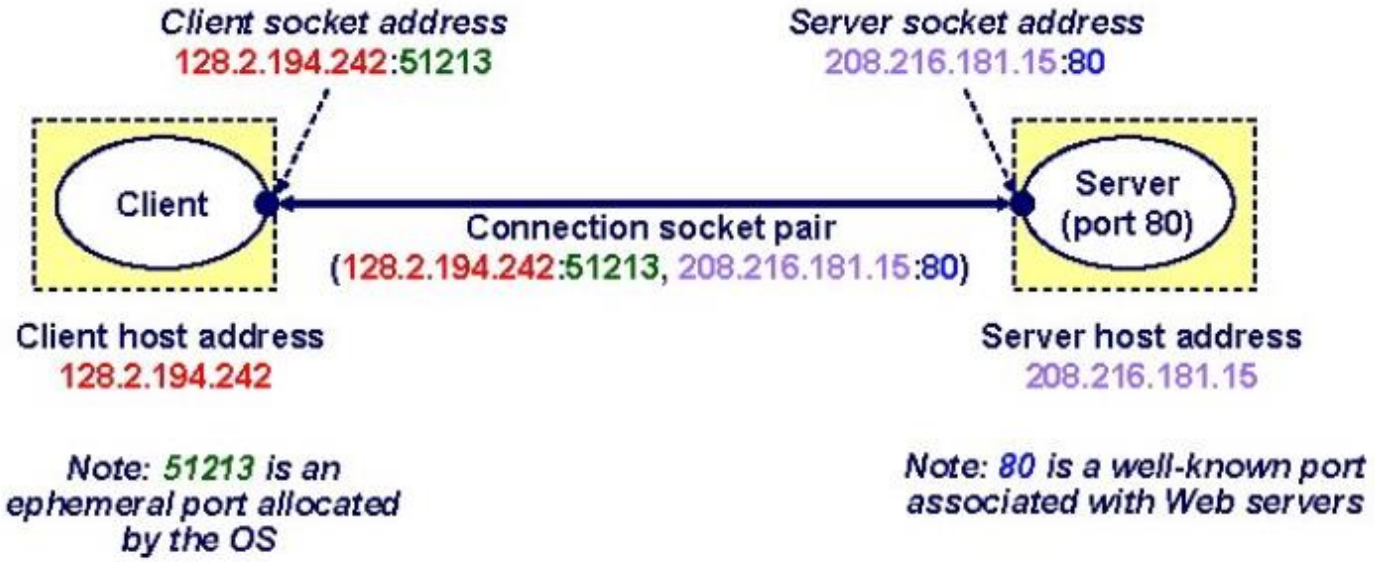
A nivel de capa de transporte cada socket tiene un **socket address**

¿Qué es un socket address?

Es la combinación entre una **dirección IP** y un **puerto**

Sockets

- A *socket* is one endpoint of a two-way communication link between two processes
  - a combination of an IP address and a port number
- Processes send/receive messages through sockets



Si vamos a comunicar dos procesos, tendremos dos sockets, uno para cada proceso

Entonces, una conexión se identifica de manera única mediante un par de socket address:

(connectorIP:connectorPuerto, listenerIP:listenerPuerto)

Para manejar y usar los sockets necesitamos un API: cada OS ofrece una

POSIX SOCKETS

Para crear el socket necesito:

Tres atributos: dominio, tipo y protocolo

Valores típicos para el Dominio

Domain	Communication performed	Communication between applications	Address format	Address structure
AF_UNIX	within kernel	on same host	pathname	*sockaddr_un*
AF_INET	via IPv4	on hosts connected via an IPv4 network	32-bit IPv4 address + 16-bit port number	*sockaddr_in*
AF_INET6	via IPv6	on hosts connected via an IPv6 network	128-bit IPv6 address + 16-bit port number	*sockaddr_in6*

Valores para el Tipo:

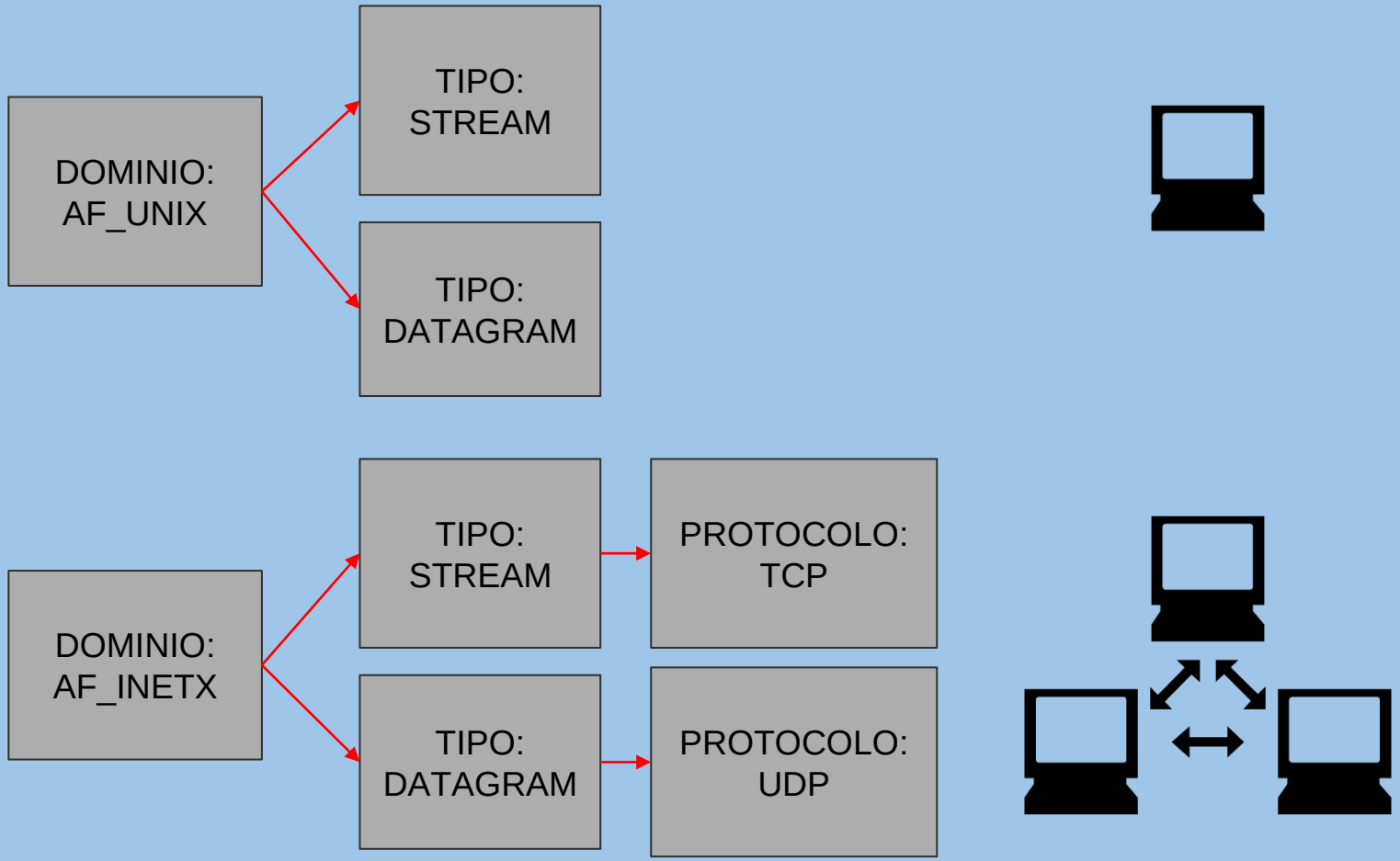
SOCK_STREAM

SOCK_DGRAM

Property	Socket type	
	Stream	Datagram
Reliable delivery?	Y	N
Message boundaries preserved?	N	Y
Connection-oriented?	Y	N

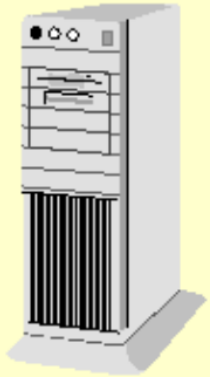
Valores para el Protocolo: colocamos **0**, el OS elige el protocolo automáticamente *(ver: man socket)

En resumen:

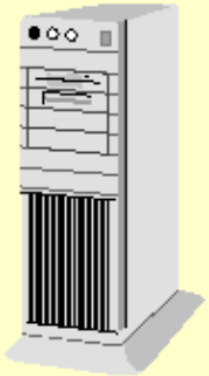

DOMINIO:
AF_UNIX
TIPO:
STREAM
TIPO:
DATAGRAM
DOMINIO:
AF_INETX
TIPO:
STREAM
PROTOCOLO:
TCP
TIPO:
DATAGRAM
PROTOCOLO:
UDP

Comunicación usando sockets

Servidor



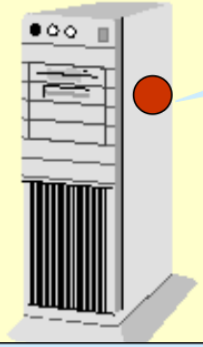
Cliente



Comunicación usando sockets

Ambos crean un *socket* para comunicarse a través de la red

Servidor



Protocolo de transporte que usará el socket

nte

descriptor = socket (protofamily, type, protocol)

0: protocolo por defecto

¿Qué es un socket descriptor?

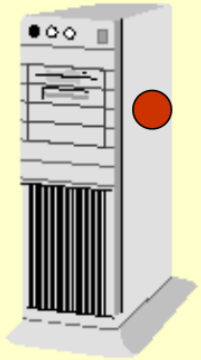
Cada socket es identificado en el kernel por medio de un entero: socket descriptor

Aunque un **socket descriptor** es diferente a un **file descriptor**, al usar el **API** de **programación**, se usan de la misma manera.

Comunicación usando sockets

El servidor asocia el descriptor del socket al **puerto** por el que recibirá las peticiones y a la **dirección local**

Servidor



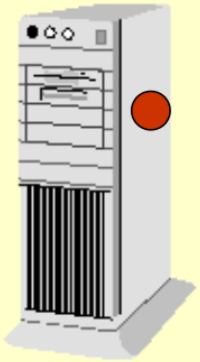
Largo de la dirección local (medido en bytes)

bind (descriptor, dir_local, largo_dir_local)

Comunicación usando sockets

Servidor

El servidor queda en espera...

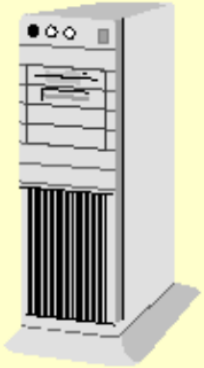


Especifica el largo de la fila de peticiones de clientes esperando ser atendidas

listen(descriptor, largo_cola)

Comunicación usando sockets

Servidor



Resumen actividades hechas por el servidor (hasta ahora):

- Crea un socket (*socket*)
- Le asocia un puerto y una dirección local (*bind*)
- Queda en espera (*listen*)

Listening process:

1. Crear un objeto de tipo **socket**, con la función socket.
2. Usar la función **bind** para conectar el socket con un EndPoint (que puede ser una IP + puerto para un AF_INET o un PATH a un **socket file** para un AF_UNIX). Es decir, le pido al kernel que asocie un socket a un socket address (AF_INET) específico o a un archivo específico (AF_UNIX)
3. Configurar el socket para escuchar, usando la función **listen**.
4. Aceptar conexión con la función **accept**.

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

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

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int bind(int sockfd, const struct sockaddr *addr,
                socklen_t addrlen);
```

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int listen(int sockfd, int backlog);
```

```
SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

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

Nota para el paso 3

En el paso 3, la función **listen** sirve para crear un **backlog**. Un backlog es una lista de conexiones pendientes que aún no han sido aceptadas por el proceso que escuchará. El sistema operativo mantendrá las conexiones hechas al proceso en backlog hasta que éste las acepte. Sí el backlog se llena, el OS comenzará a rechazar las conexiones.

El backlog es una cola. Si esta es pequeña, se rechazarán muchas conexiones, pero sí es muy grande, es posible que se presenten timeouts y se desconecten los procesos que están esperando a ser aceptados.

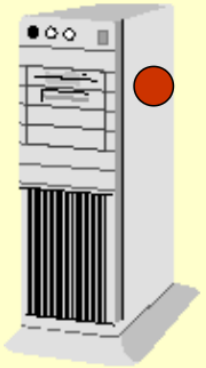
Nota para el paso 4

- Después del paso 3 (el backlog queda configurado), para cada conexión entrante, se debe llamar la función accept. Aquí se podría tener un hilo en ciclo infinito para cada conexión.
- Cada llamado a la función accept, sacará del backlog una conexión en espera.
- Sí el backlog está vacío y el listener socket se configura para bloquearse, el proceso se bloqueará hasta que no entre una nueva conexión.
- La función accept retorna un objeto socket que representa la conexión con el connector process.

Comunicación usando sockets

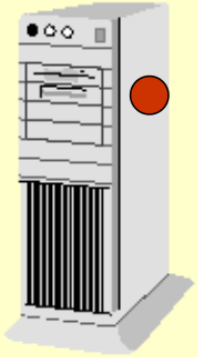
Y el cliente... una vez creado el socket...

Cliente



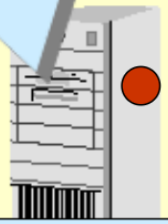
Comunicación usando sockets

Servidor



Se conecta (sólo si se trata de un servicio SOCK_STREAM) al servidor a través de su socket. Para esto le asocia el puerto del servidor y su dirección .

ente



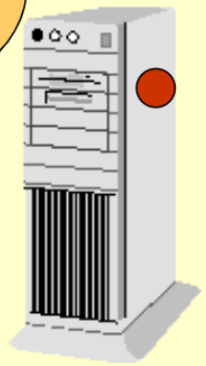
connect (descriptor, dir_destino, largo_dir_destino)

Comunicación usando sockets

Resumen actividades hechas por el cliente (hasta ahora):

- Crea un socket (***socket***)
- Le asocia un puerto y una dirección destino (***connect,*** sólo en caso de un servicio con conexión)

Cliente

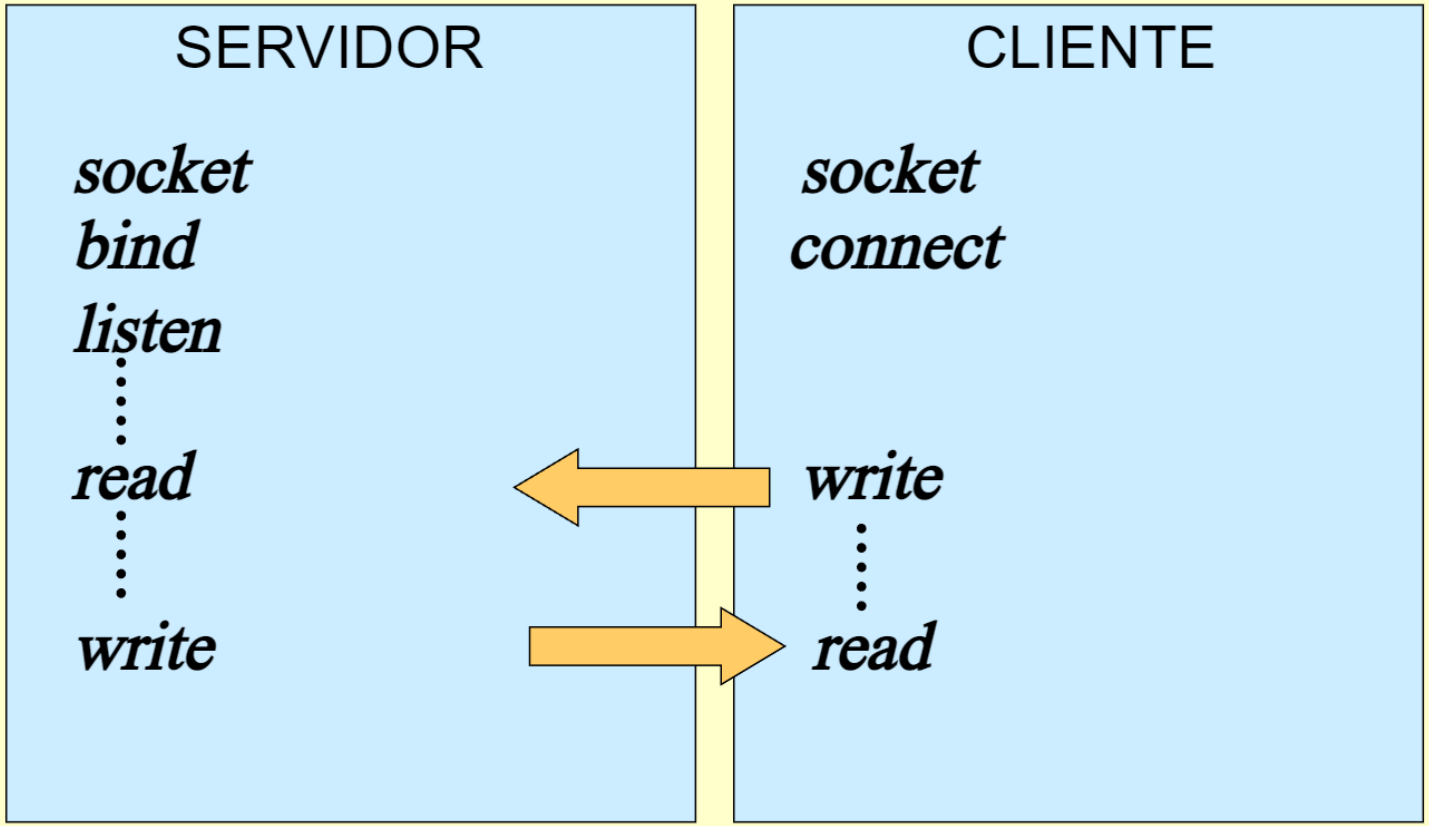


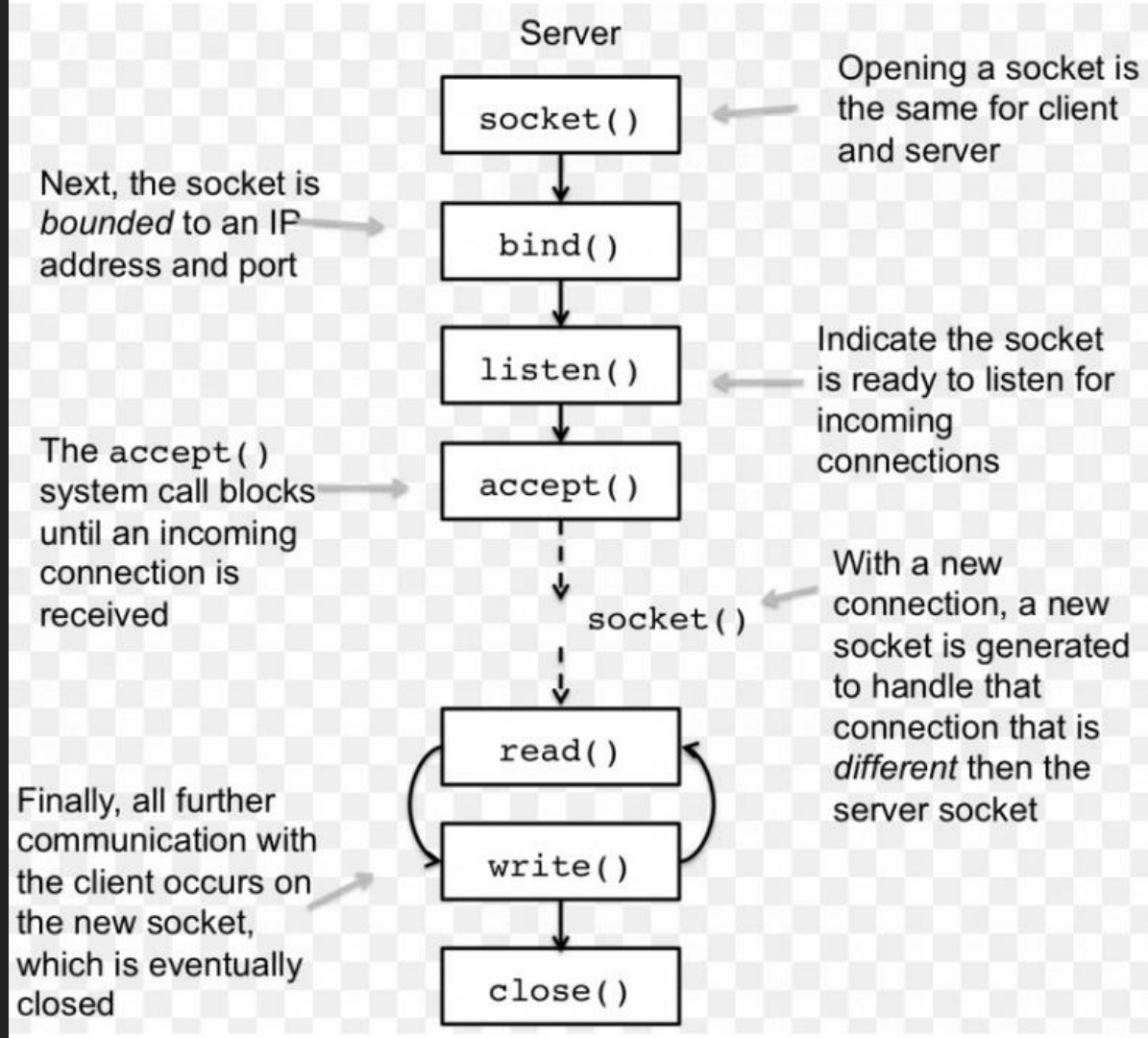
Connector process

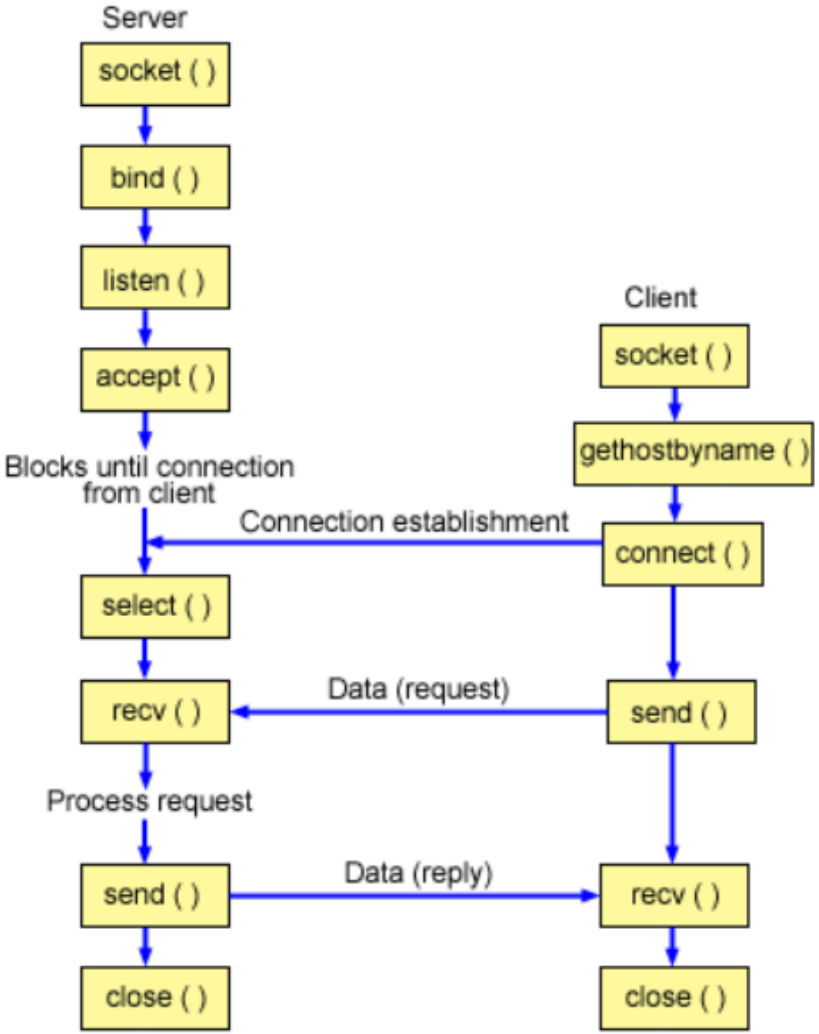
1. El listening process ya debe estar en modo listening (con el backlog creado).
2. Llamar la función socket para **crear un socket** object.
3. Llamar la función **connect** pasando como parámetro el EndPoint del listening process. SÍ la función es exitosa, quiere decir que el listening process aceptó la conexión. Antes de esto, el connector process estará en el backlog esperando a ser aceptado.

NOTA: connect retornará un socket descriptor (un entero que identifica al socket en el kernel) que servirá para comunicarse con el listening process.

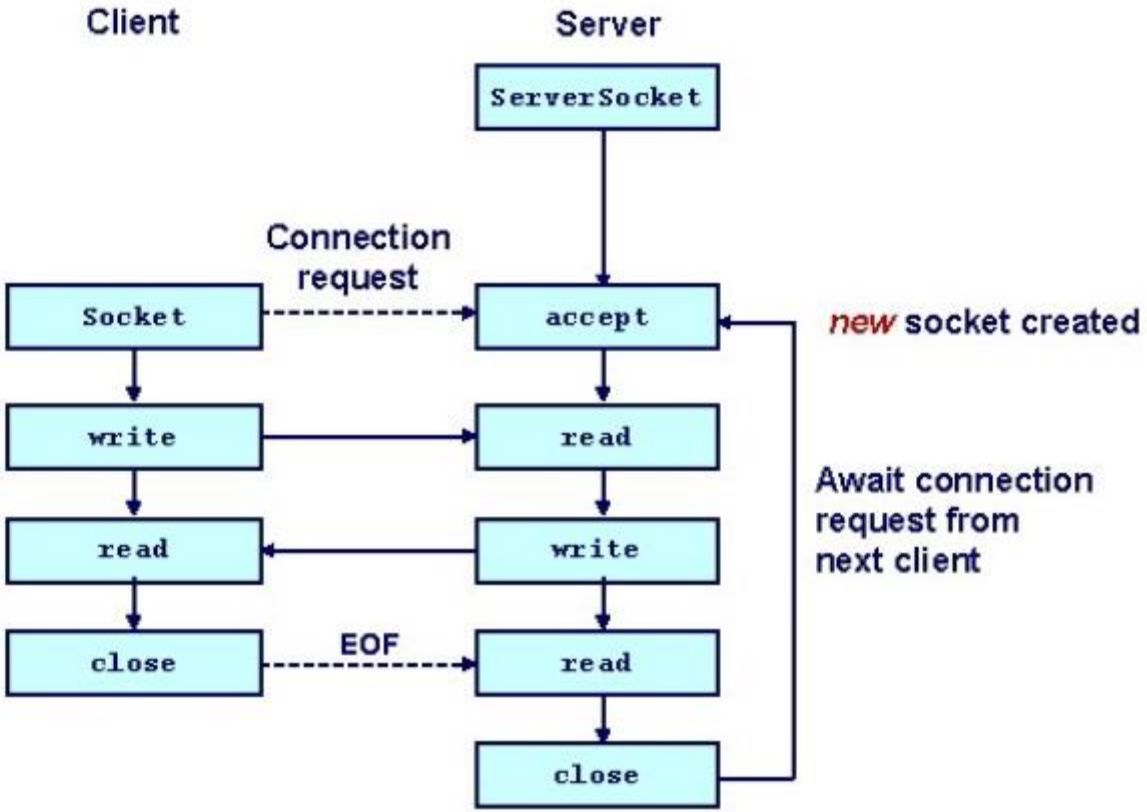
Comunicación usando sockets
RESUMEN




Server
socket()
Opening a socket is the same for client and server
Next, the socket is *bounded* to an IP address and port
bind()
listen()
Indicate the socket is ready to listen for incoming connections
The accept() system call blocks until an incoming connection is received
accept()
socket()
With a new connection, a new socket is generated to handle that connection that is *different* then the server socket
read()
write()
Finally, all further communication with the client occurs on the new socket, which is eventually closed
close()


Server
socket ( )
bind ( )
listen ( )
accept ( )
Blocks until connection from client
select ( )
recv ( )
Process request
send ( )
close ( )
Client
socket ( )
gethostbyname ( )
Connection establishment
connect ( )
Data (request)
send ( )
Data (reply)
recv ( )
close ( )

Java Sockets Interface



Help with C Sockets Interface