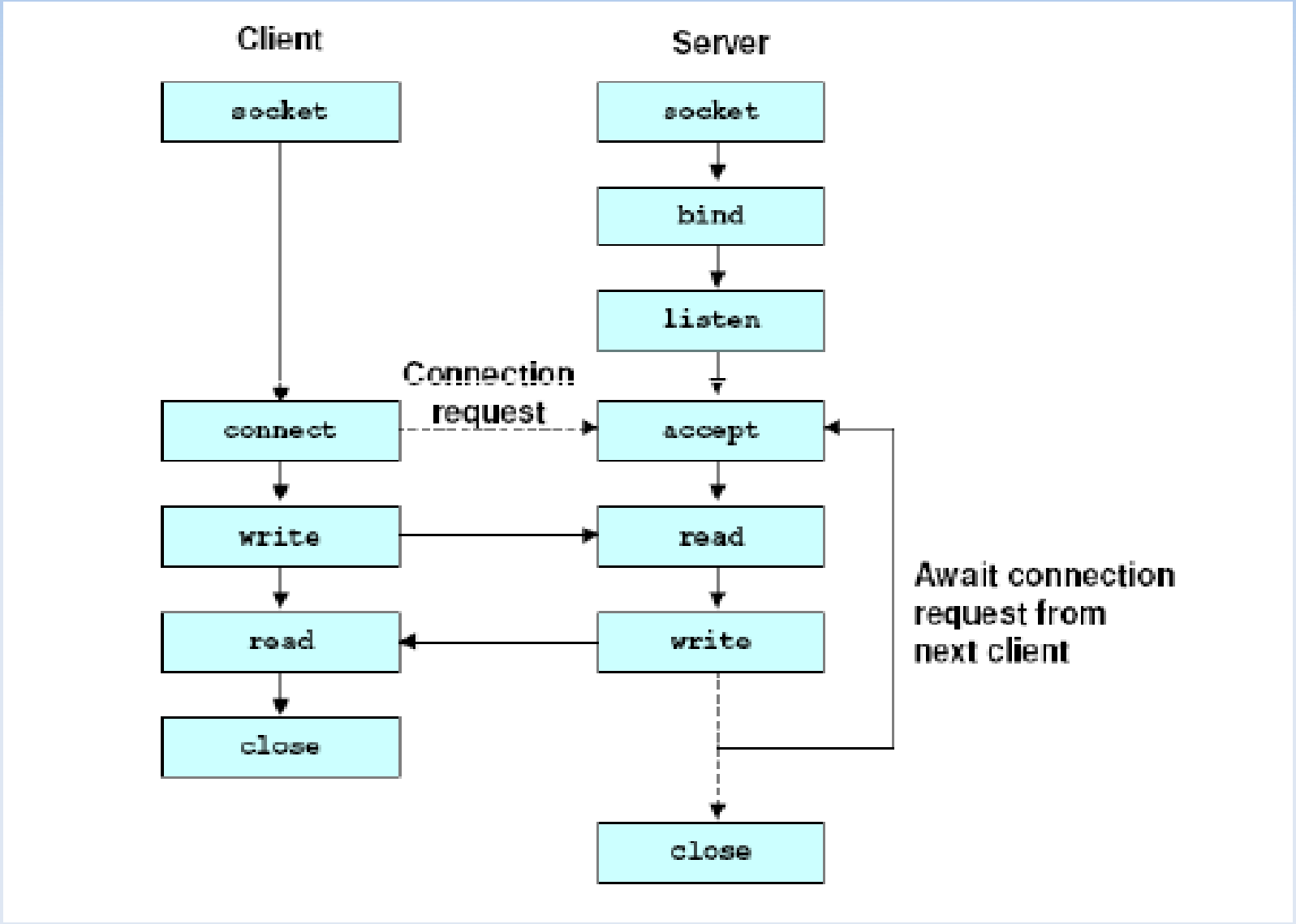
API de Sockets



Objetivo

- **Comprender el uso de la de la Interfaz de Aplicaciones (API) Sockets de Berkeley.**

API de Sockets

- Fue introducida en BSD4.1 UNIX, 1981
- El socket es explícitamente creado, usado, y liberado por las aplicaciones
- Sigue el modelo cliente/servidor
- Hay dos tipos de servicios de transporte vía el API de socket:
  - Datagramas no confiables (usando UDP)
  - Orientado a un flujo de bytes y confiable (usando TCP)

- Los sockets son estructuras de datos locales al host
  - Son creados por la aplicación,
  - Ofrecen una interfaz controlada por el OS (una "puerta") a través de la cual el proceso aplicación puede tanto enviar como recibir mensajes a/desde el otro proceso aplicación

Conceptos de Diseño

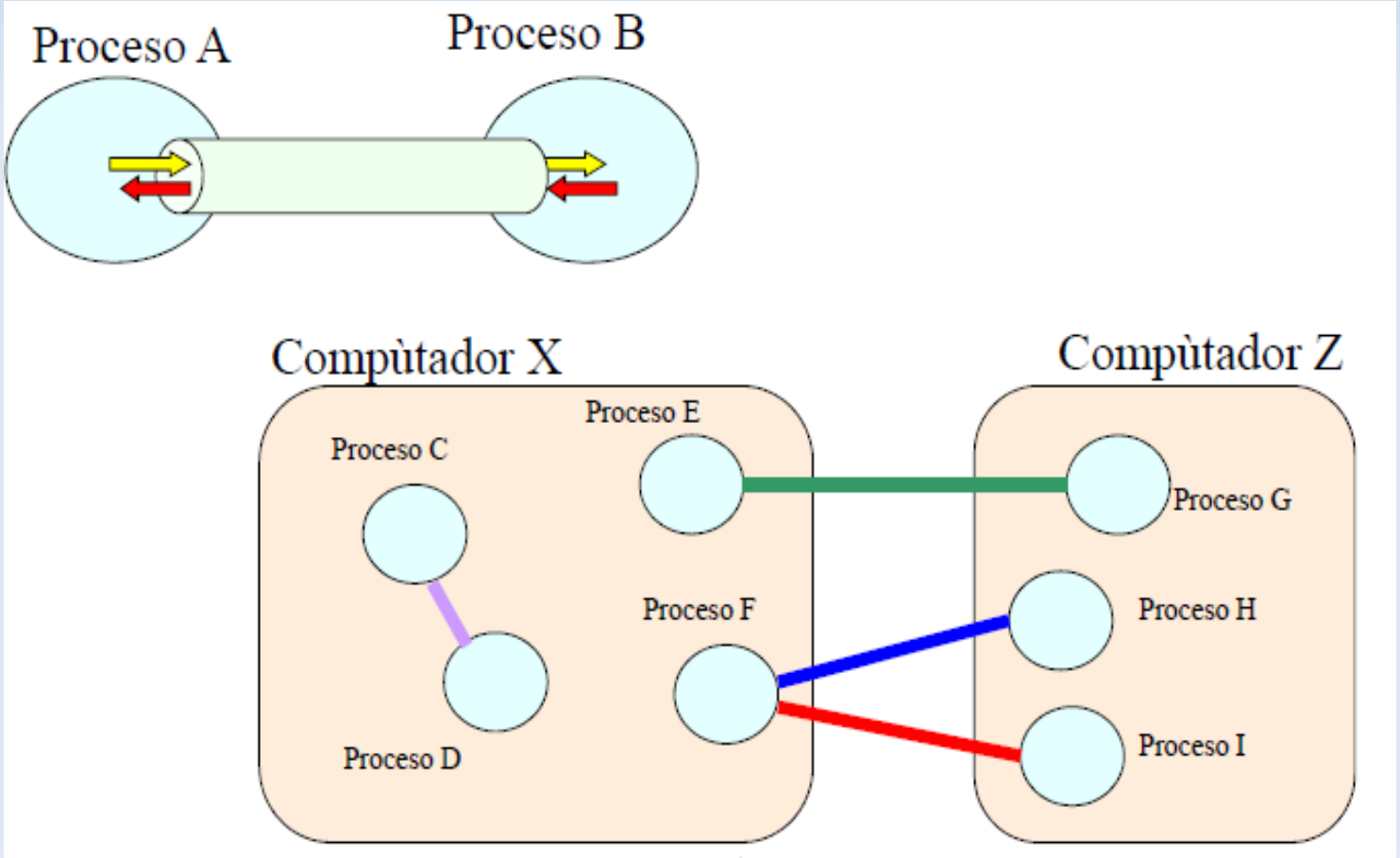
- Para que dos procesos se pudieran comunicar hubo que dotar al sistema de una serie de funciones que permitieran a esos procesos acceder a los dispositivos.
- Cuando se consideró cómo añadir funciones al sistema operativo para suministrar acceso a las comunicaciones, surgieron dos posibilidades:

Conceptos de Diseño

- Definir funciones que soportaran específicamente el protocolo TCP/IP.
- Definir funciones que soportaran cualquier tipo de protocolo de comunicaciones y parametrizarlas cuando se quisiera utilizar TCP/IP.
- Los diseñadores optaron por la segunda opción: mantener la generalidad del interfaz

Definición

- Un socket, desde el punto de vista funcional, se define como un punto terminal al que pueden “enchufarse” dos procesos para comunicar entre sí.



Sockets

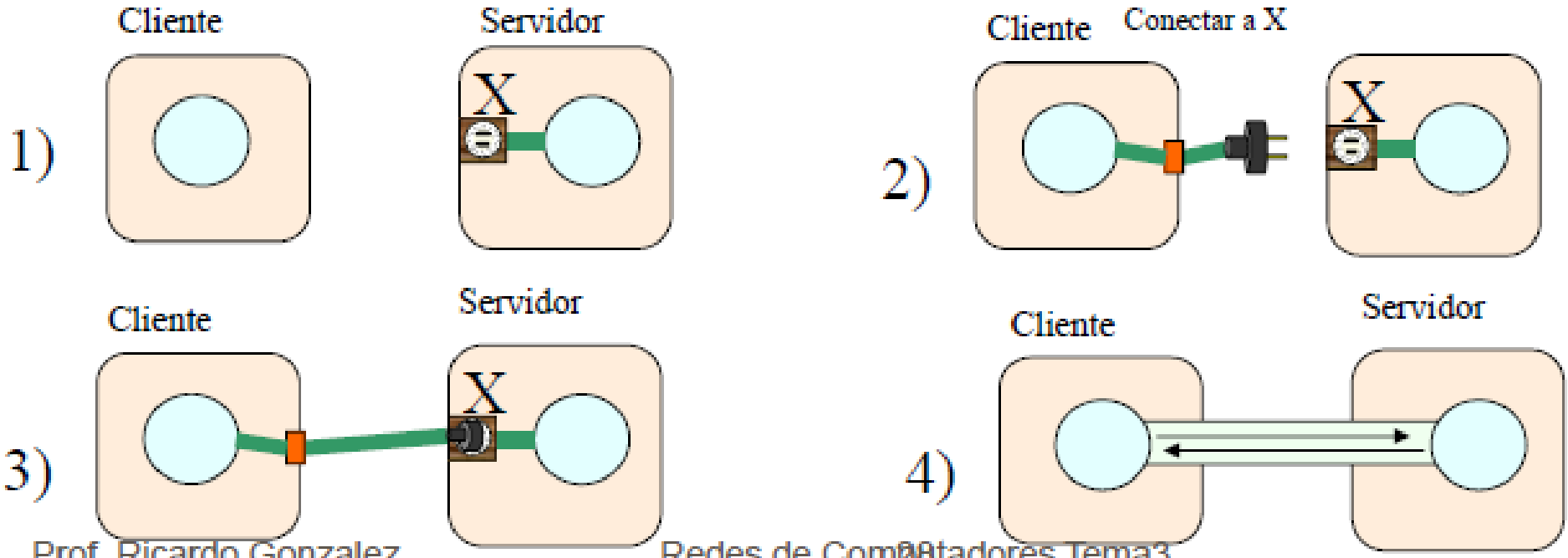
Los pasos, en general para comunicarse vía sockets son:

1) El servidor crea un socket cuyo nombre conocen otros procesos. Abre una conexión a un puerto bien conocido. Un puerto es un concepto lógico para saber a qué proceso dirigir la petición del cliente.

2) El cliente crea un socket sin nombre y pide una conexión al socket del servidor (con el puerto bien conocido).

3) El servidor acepta la conexión.

4) El cliente y el servidor intercambian información.



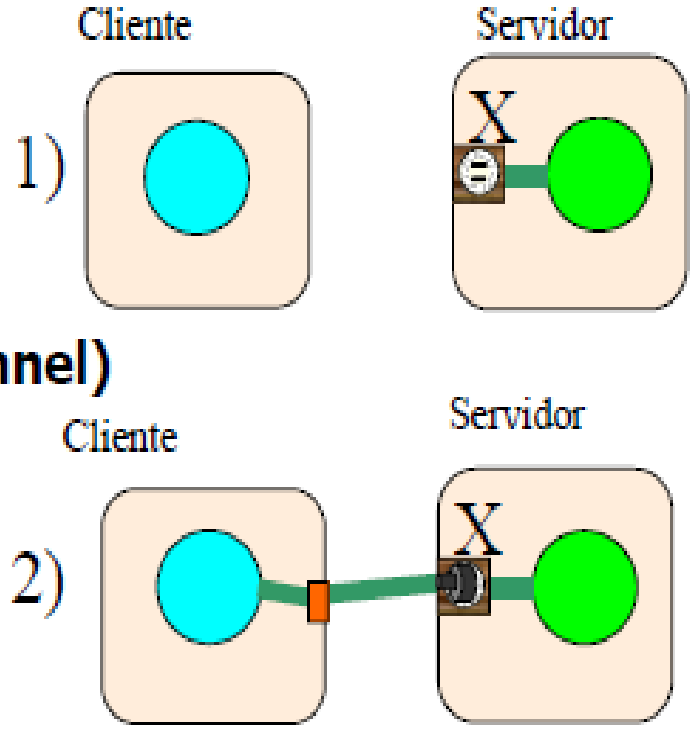
Tipos de Servidores: Seriales (TCP)

- **Seriales:** reciben la petición y hasta que no terminan con la misma no atienden una nueva petición.

```
for(;;) {
        listen for client request
        create a private two-way channel
        while (no error on communication channel)
            read request
            handle request
        close communication channel
}
```

1

2

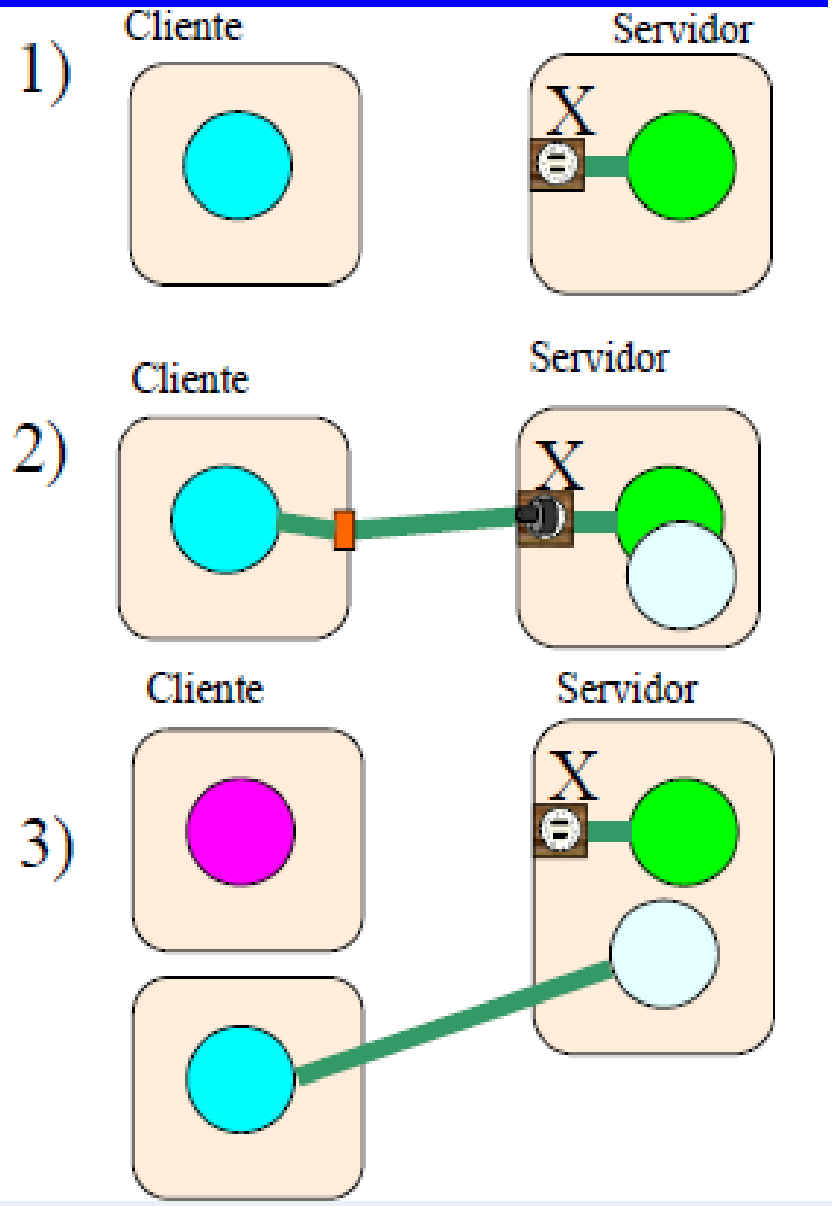


Si el servidor es muy solicitado y las peticiones son largas (transferencia de archivos) los clientes tendrán que esperar para ser atendidos lo cual puede llegar a ocasionar inconvenientes.

Tipos de Servidores: Concurrentes (TCP)

Padre-Hijo: (Servidor concurrente) el hijo maneja la petición del cliente mientras el padre sigue oyendo requerimientos adicionales.

```
for(;;) {
    listen for client request        1
    create a private two-way          2
     channel
    if (!fork)
        handle the request            3
        exit
    else
        close the private channel
    clean up zombies
}
```



Esquema de la comunicación Cliente / Servidor (TCP)

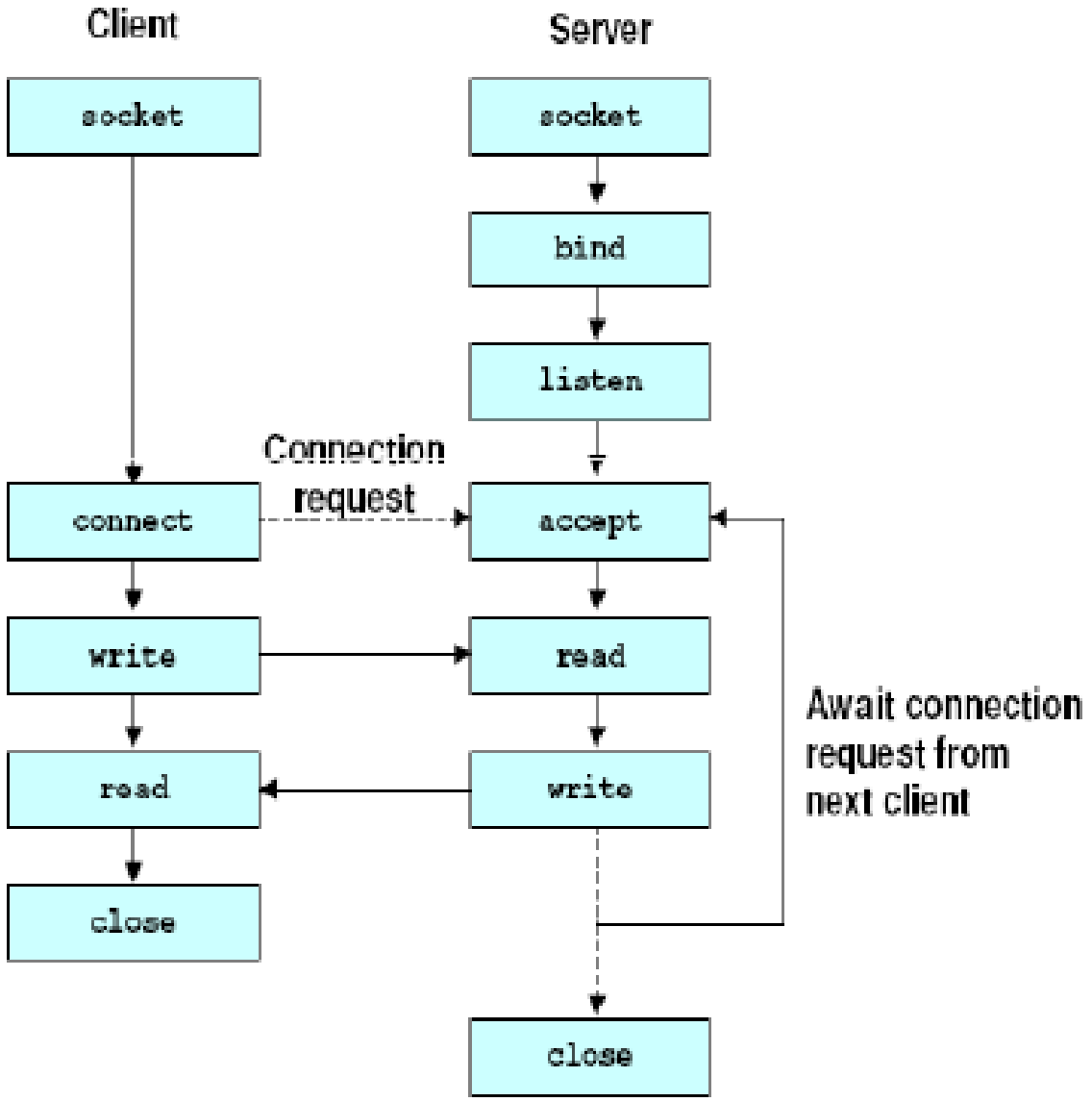
Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   1. Accept new connection
   2. Communicate
   3. Close the connection

Esquema de la comunicación Cliente / Servidor (TCP)



Uso de los Sockets

1. Creación del *socket* (mediante la función socket()).

2. Asignar una dirección final al *socket*, una dirección a la que pueda referirse el otro interlocutor. Esto se hace con la función bind().

Uso de los Sockets

3. En los *sockets* tipo *stream*, es necesario conectar con otro *socket* cuya dirección debemos conocer. Esto se logra ejecutando la función connect() para el proceso que actuará de cliente y las funciones listen() y accept() para el proceso que actuará como servidor.

Uso de los sockets

4. Comunicarse. Esta es la parte más sencilla. En los *sockets* tipo *stream*, basta usar write() para volcar datos en el *socket* que el otro extremo puede leer mediante la función read(), y a la inversa. En los *sockets* tipo *datagram*, el envío se realiza con la función sendto() y la recepción se realiza con recvfrom(). Ambas deben especificar siempre la dirección final del otro interlocutor.

Uso de los Sockets

5. Finalmente, cuando ya la comunicación se da por finalizada, ambos interlocutores deben cerrar el *socket* mediante la función close() o shutdown().

API de Sockets

- Los *sockets* son un API para la comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando en distintas máquinas.
- Un **socket** es al sistema de comunicación entre computadores lo que un buzón o un teléfono es al sistema de comunicación entre personas: un punto de comunicación entre dos agentes (procesos o personas respectivamente) por el cual se puede emitir o recibir información.
- La forma de referenciar un socket por los procesos implicados es mediante un **descriptor** del mismo tipo que el utilizado para referenciar archivos.
- Se podrá realizar redirecciones de los archivos de E/S estándar (descriptores 0,1 y 2) a algun sockets y así combinar aplicaciones de la red.
- Todo nuevo proceso creado heredará, los descriptores de archivos y los sockets de su padre.

Referencias Bibliográficas

- Douglas Comer & David Stevens. Internetworking with TCP/IP. Volume III. Cliente – Server Programming and Applications. Prentice-Hall. 1993.
- W. R. Stevens. *UNIX Network Programming*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
- W. R. Stevens. *UNIX Network Programming*. Prentice-Hall, Englewood Cliffs, New Jersey, second edition, 1998.~