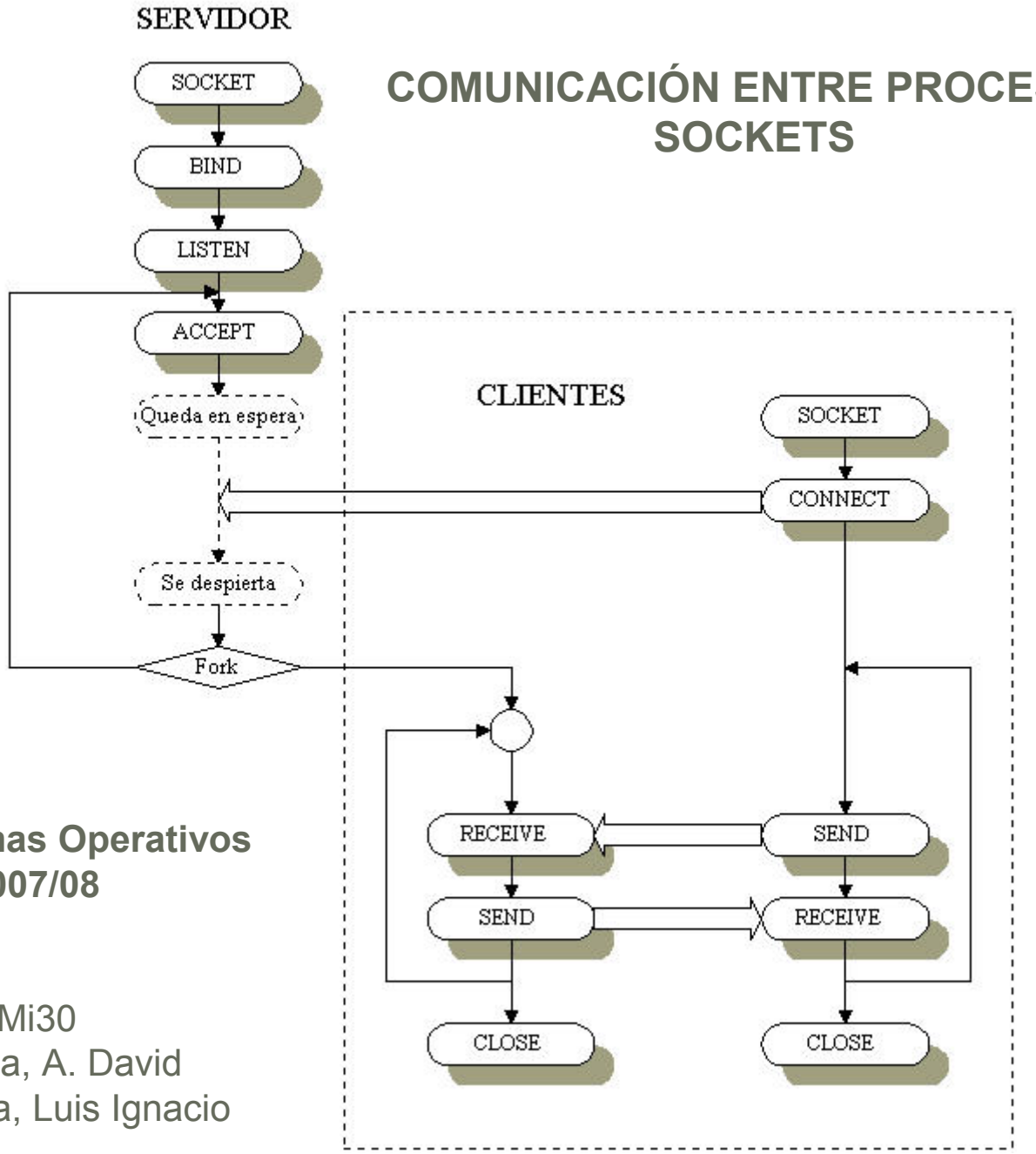
COMUNICACIÓN ENTRE PROCESOS
SOCKETS



Diseño de Sistemas Operativos
Curso 2007/08

Grupo: Mi30
Martín Santana, A. David
Martínez Santana, Luis Ignacio

¿Qué son?(Definición)

- Los ***sockets*** son mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando en distintas máquinas.
- Una forma de conseguir que dos programas se transmitan datos.
- Un **socket** no es más que un "canal de comunicación" entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador.
- Desde el punto de vista de programación, un **socket** no es más que un "fichero" que se abre de una manera especial.

¿Qué son? (Características)

- Una vez abierto se pueden escribir y leer datos de él con las funciones de **read()** y **write()**.
- La forma de referenciar un socket por los procesos implicados, es mediante un **descriptor** del mismo tipo que el utilizado para referenciar ficheros.
- Se podrá realizar redirecciones de los archivos de E/S estándar (descriptores 0,1 y 2) a los sockets y así combinar entre ellos aplicaciones de la red.

Propiedades

- Fiabilidad de la Transmisión. No se pierden los datos transmitidos.
- Conservación del Orden de los Datos. Los datos llegan en el orden en que se emitieron.
- No Duplicación de los Datos. El Dato sólo llega una vez.
- Comunicación en modo conectado. La conexión está establecida antes de iniciar la comunicación. De este modo, la emisión desde un extremo va destinada al otro (implícitamente).
- Conservación de los límites de los mensajes. Los límites de mensajes emitidos pueden encontrarse o conocerse en el destino.
- Envío de Mensajes "urgentes". Permite el envío de datos fuera de flujo o fuera de banda. Al enviar datos fuera del flujo normal, son accesibles de inmediato.

Atributos

Un socket se caracteriza por tres atributos:

- **Dominio**: Especifica el medio de comunicación de la red que el socket utilizará.
- **Protocolo**: Especifica que protocolo se va a usar.
- **Tipo**: Los protocolos de internet proveen dos niveles distintos de servicio : flujo y datagramas.

Atributos (Dominio)

- **AF_UNIX**: Sockets internos de UNIX
  (Sockets del sistema de archivos).
- **AF_INET**: Protocolos de internet ARPA
  (Sockets de redes de UNIX).
- **AF_ISO**: Protocolos estándar ISO.
- **AF_NS**: Protocolos de redes Xerox

Atributos (Protocolo)

- Se usa donde el mecanismo de transporte permite más de un protocolo a ser usado.
- En las redes de UNIX y en los sockets de sistema de archivos no necesitamos seleccionar otro protocolo diferente al default.

Atributos(Tipo)

SOCK_STREAM: Para flujo

- Son implementados en el dominio AF_INET por conexiones TCP/IP.
- Son el tipo usual en el dominio AF_UNIX.

SOCK_DGRAM : Para datagramas

- No establecen ni mantienen una conexión.
- También existe un límite en el tamaño del datagrama que se puede enviar.
- Se transmite como un solo mensaje en la red que se puede perder, duplicar o llegar fuera de secuencia.

Tipos de Sockets

Existen básicamente dos tipos**:**

- Los **no orientados a conexión**
  - El programa de aplicación da la fiabilidad
- Los **orientados a conexión**.
  - Comunicaciones fiables
  - Circuito Virtual

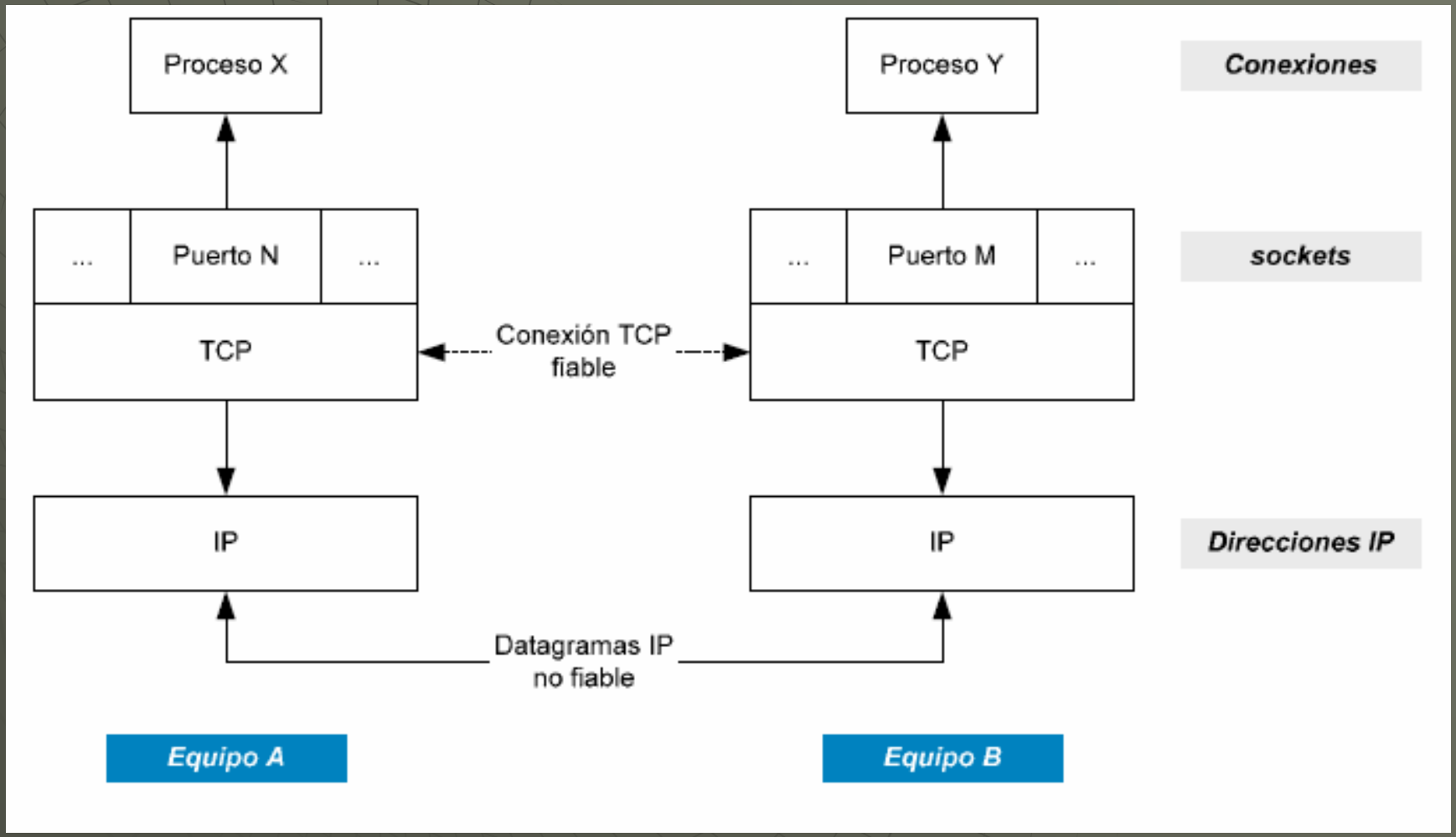
Tipos de Sockets

- Un ***socket*** queda definido por una dirección IP, un protocolo y un número de puerto.

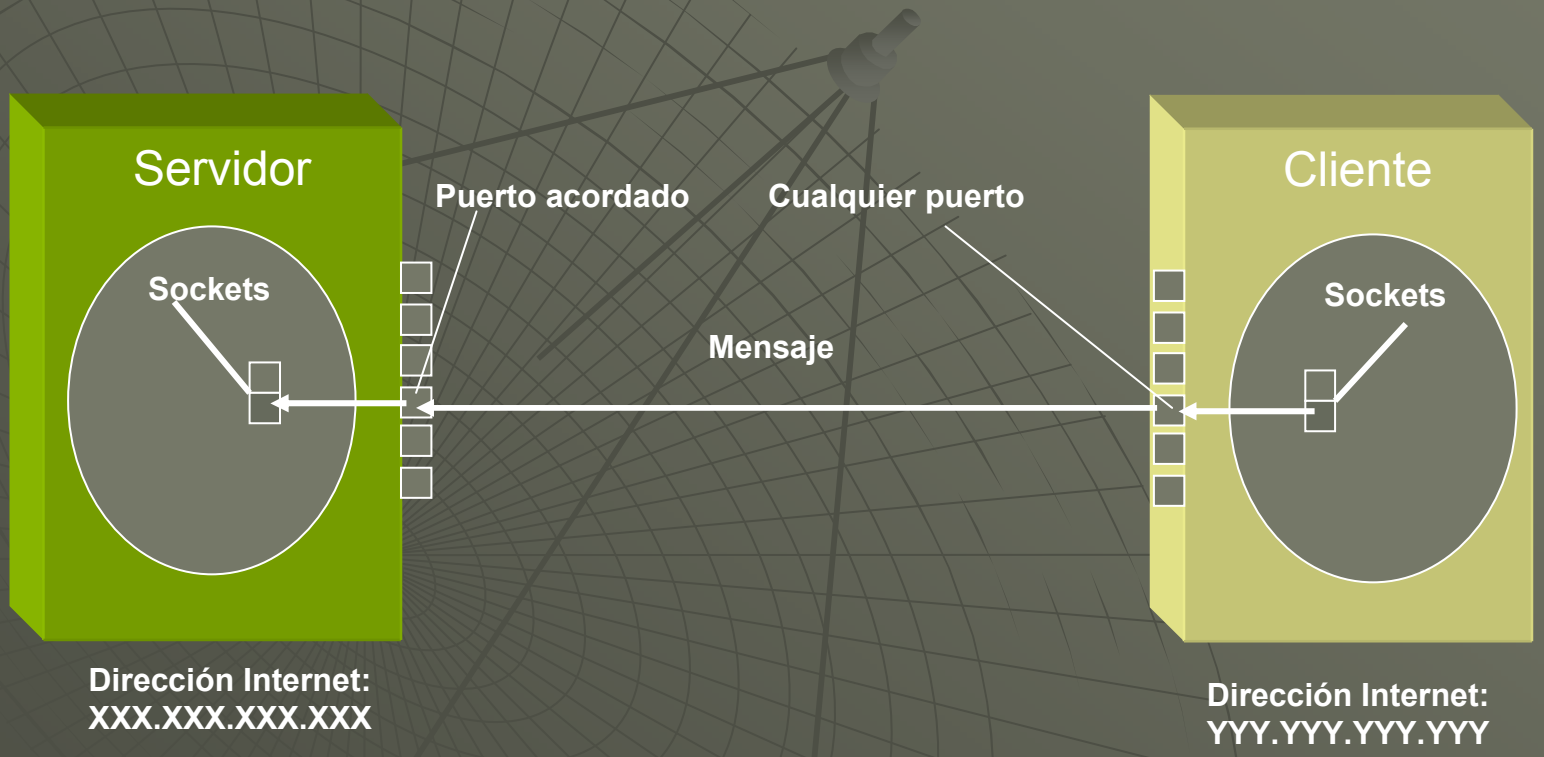
- En el caso concreto de TCP-IP, un socket se define por una dupla Origen – Destino.

- Tanto el origen como el destino vienen indicados por un par (ip, puerto).

Tipos de Sockets (Diagrama)



Tipos de Sockets(Puertos)



Sockets no orientado a conexión

- Es el llamado protocolo **UDP.**
- No es necesario que los programas se conecten.
- Cualquiera de ellos puede transmitir datos en cualquier momento, independientemente de que el otro programa esté "escuchando" o no.
- Garantiza que los datos que lleguen son correctos, pero no garantiza que lleguen todos.
- Se utiliza cuando es muy importante que el programa no se quede bloqueado.
- No importa que se pierdan datos.

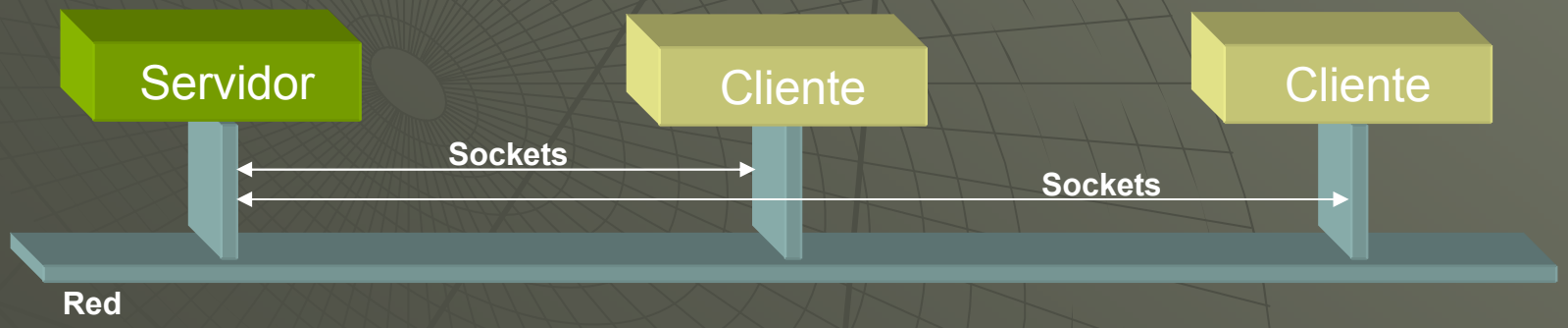
Sockes orientado a conexión.

- Primero hay que establecer correctamente la conexión.
- Ninguno de los dos puede transmitir datos.
- Se usa el protocolo **TCP** del protocolo **TCP/IP**, para gestionar la conexión.
- Se garantiza que todos los datos van a llegar de un programa al otro correctamente.
- Se utiliza cuando la información a transmitir es importante, no se puede perder ningún dato.
- No importa que los programas se queden "bloqueados" esperando o transmitiendo datos.

Arquitectura de Conexión

Cliente – Servidor

- **Servidor:** es el programa que permanece pasivo a la espera de que alguien solicite conexión con él.
- **Cliente:** es el programa que solicita la conexión para pedir datos al servidor



Arquitectura de la conexión.

- **Servidor:**
  - Está ejecutándose y esperando a que otro quiera conectarse a él.
  - Nunca da "el primer paso" en la conexión.
  - Es el que "sirve" información al que se la pida.
- **Cliente:**
  - Es el programa que da el “primer paso” en la conexión.
  - En el momento de ejecutarlo o cuando lo necesite, intenta conectarse al servidor.
  - Es el que solicita información al **servidor**.

Conexión

- Para poder realizar la conexión entre ambos programas (cliente y servidor) es necesario conocer:
  - **Dirección IP** del servidor.
  - **Servicio** que queremos crear / utilizar.

Servidor (Funciones)

- Funciones del servidor en **modo conectado:**
- Las funciones utilizadas son las siguientes:
  - int **socket** (int *dominio*, int *tipo*, int *protocolo*)
  - int **bind** (int *dfServer*, struct sockaddr* *direccServer*, int *longDirecc*)
  - int **listen** (int *dfServer*, int *longCola*)
  - int **accept** (int *dfServer*, struct sockaddr* *direccCliente*, int* *longDireccCli*)

Servidor (Funciones)

- int **socket** (int *dominio*, int *tipo*, int *protocolo*)
  - crea un socket sin nombre de un dominio, tipo y protocolo específico
  - *dominio* : AF_INET, AF_UNIX
  - *tipo* : SOCK__DGRAM, SOCK__STREAM
  - *protocolo* : 0 (protocolo por defecto)

- int **bind** (int *dfServer*, struct sockaddr* *direccServer*, int *longDirecc*)
  - nombra un socket: asocia el socket no nombrado de descriptor *dfServer* con la dirección del socket
  - almacenado en *direccServer*.
  - La dirección depende de si estamos en un dominio AF_UNIX o AF_INET.

Servidor (Funciones)

- int **listen** (int *dfServer*, int *longCola*)
  - especifica el máximo número de peticiones de conexión pendientes.

- int **accept** (int *dfServer*, struct sockaddr* *direccCliente*, int* *longDireccCli*)
  - escucha al socket nombrado "servidor *dfServer"* y espera hasta que se reciba la petición de la conexión de un cliente. Al ocurrir esta incidencia, crea un socket sin nombre con las mismas características que el socket servidor original, lo conecta al socket cliente y devuelve un descriptor de fichero que puede ser utilizado para la comunicación con el cliente.

Servidor (Funcionalidad)

- El programa servidor realiza los siguientes pasos:

- **Apertura de un socket**, mediante la función **socket()**.
  - Esta función devuelve un descriptor de fichero normal, como puede devolverlo open().
  - La función socket() no hace absolutamente nada, salvo devolvernos y preparar un descriptor de fichero que el sistema posteriormente asociará a una conexión en red.

- **Avisar al sistema operativo** de que hemos abierto un socket y queremos que asocie nuestro programa a dicho socket.
  - Se consigue mediante la función **bind()**.
  - El sistema todavía no atenderá a las conexiones de clientes, simplemente anota que cuando empiece a hacerlo, tendrá que avisarnos a nosotros. Es en esta llamada cuando se debe indicar el número de servicio al que se quiere atender.

Servidor (Funcionalidad)

- Avisar al sistema de que **comience a atender dicha conexión** de red.
  - Se consigue mediante la función **listen()**.
  - A partir de este momento el sistema operativo anotará la conexión de cualquier cliente para pasárnosla cuando se lo pidamos.
  - Si llegan clientes más rápido de lo que somos capaces de atenderlos, el sistema operativo hace una "cola" con ellos y nos los irá pasando según vayamos pidiéndolo.
- Pedir y **aceptar las conexiones** de clientes al sistema operativo.
  - Para ello hacemos una llamada a la función **accept()**.
  - Esta función le indica al sistema operativo que nos dé al siguiente cliente de la cola.
  - Si no hay clientes se quedará bloqueada hasta que algún cliente se conecte.

Servidor (Funcionalidad)

- **Escribir y recibir datos** del cliente, por medio de las funciones **write()** y **read()**, que son exactamente las mismas que usamos para escribir o leer de un fichero.
  - Obviamente, tanto cliente como servidor deben saber qué datos esperan recibir, qué datos deben enviar y en qué formato.
  - Puedes ver cómo se pueden poner de acuerdo en estos mensajes en el apartado de mensajes.
- **Cierre de la comunicación** y del socket, por medio de la función **close()**, que es la misma que sirve para cerrar un fichero.

Cliente (Funciones)

- Es el proceso encargado de crear un socket sin nombre y posteriormente enlazarlo con el socker servidor nombrado.
- Por lo tanto, es el proceso que demanda una conexión al servidor.

- Las funciones utilizadas son las siguientes:

  - int **socket** (int *dominio*, int *tipo*, int *protocolo*)

  - int **connect** (int *dfCliente*, struct sockaddr* *direccServer*, int *longDirecc*)

Cliente (Funciones)

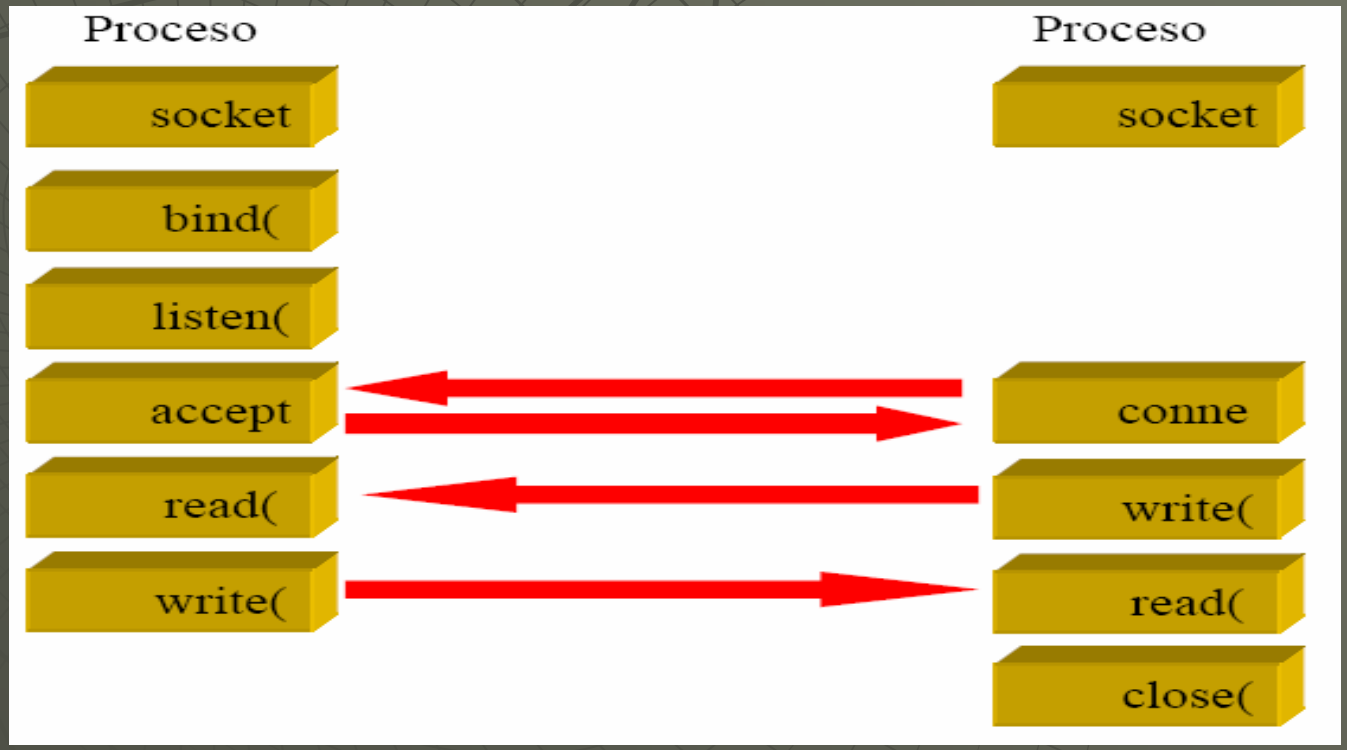
- int **socket** (int *dominio*, int *tipo*, int *protocolo*)
  - crea un socket sin nombre de un dominio, tipo y protocolo específico
  - *dominio* : AF_INET, AF_UNIX
  - *tipo* : SOCK__DGRAM, SOCK__STREAM
  - *protocolo* : 0 (protocolo por defecto)
- int **connect** (int *dfCliente*, struct sockaddr* *direccServer*, int *longDirecc*)
  - intenta conectar con un socket servidor cuya dirección se encuentra incluida en la estructura apuntada por *direccServer*.
  - El descriptor *dfCliente* se utilizará para comunicar con el socket servidor.
  - El tipo de estructura dependerá del dominio en que nos encontremos.

Cliente (Funcionalidad)

- El programa cliente realiza los siguientes pasos:
- **Apertura de un socket**, como el servidor, por medio de la función **socket()**
- **Solicitar conexión** con el servidor por medio de la función **connect()**.
  - Dicha función quedará bloqueada hasta que el servidor acepte nuestra conexión o bien si no hay servidor en el sitio indicado, saldrá dando un error.
  - En esta llamada se debe facilitar la dirección IP del servidor y el número de servicio que se desea.
- **Escribir y recibir datos** del servidor por medio de las funciones **write()** y **read()**.
- **Cerrar la comunicación** por medio de **close()**.

Ejemplos de conexión

- Los sockets orientados a conexion (TCP)
  - Sockets están orientado a ristras (stream).
  - Permite la conexión por Circuito Virtual
  - Full Duplex



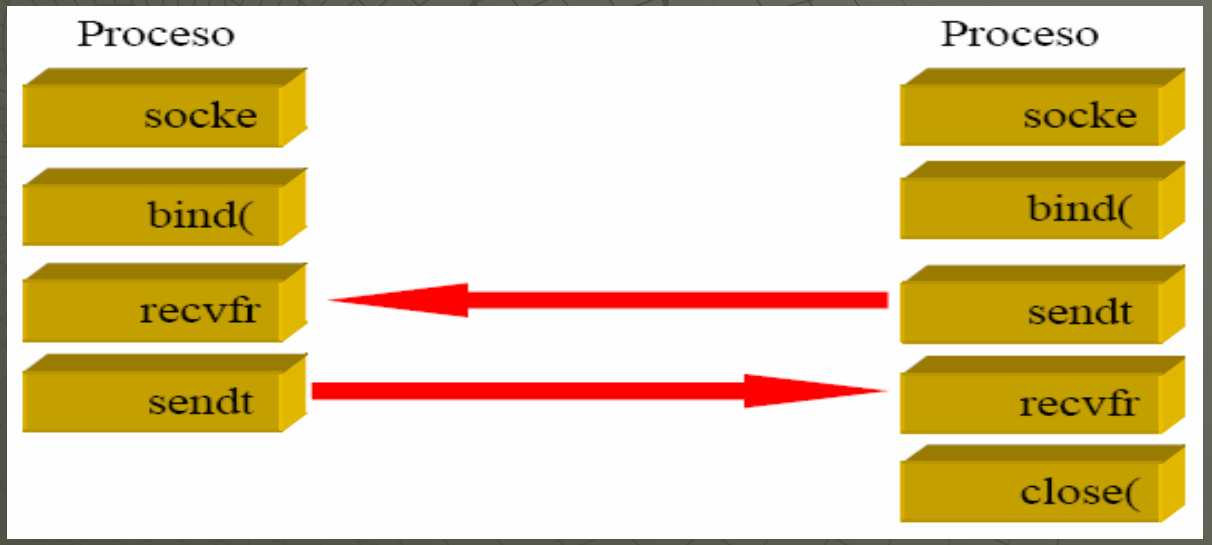
Ejemplos de conexión

- Los sockets no orientados a conexión (UDP).

–Las comunicaciones correspondientes tienen la propiedad de conservar los límites de los mensajes enviados.

–En el dominio Internet, el protocolo subyacente es el UDP.

–La transmisión es a nivel de paquetes, donde cada paquete puede seguir una ruta distinta, no garantizándose una recepción secuencial de la información.



Comparativa

Sockets	Pipes
Referenciado por descriptores	Referenciados por array de descriptores
Comunicación entre procesos de la misma y diferentes máquinas.	Comunicación sólo entre procesos de la misma máquina.
Comunicación bidireccional.	Comunicación unidireccional.
Filosofía Cliente-Servidor	Simple intercambio de información

Bibliografía y referencias

- Apuntes de la asignatura. Campus Virtual.
- TCP/IP Douglas E. Comer.
- Starlinux.net
  http://www.starlinux.net/staticpages/index.php?page=20020720164837437