Comunicación de aplicaciones

TCP/IP

Comunicación de aplicaciones

- En aplicaciones Cliente – Servidor, el servidor proporciona el servicio y el cliente es el consumidor del recurso.
- Dicha comunicación debe ser confiable.
- Ambos programas deben conectarse entre si y hasta entonces transmitir.
- TCP/IP garantiza que los datos lleguen correctamente.

TCP y UDP en TCP/IP

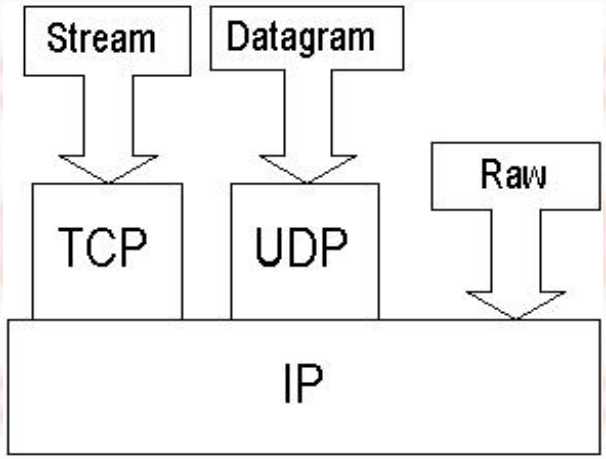
- TCP:
  - Garantiza que todos los datos van a llegar de un programa al otro correctamente.
  - La información no se debe perder aún cuando los programas se queden bloqueados o transmitiendo.
- UDP:
  - Garantiza que los datos que lleguen son correctos, pero no garantiza que lleguen todos.
  - No es tan relevante la pérdida de datos.

Comunicación de aplicaciones

- Socket: manera de hablar con otra computadora a través de descriptores de archivos estándar en Unix.
- Unix: todas las entradas y salidas son realizadas escribiendo o leyendo en un descriptor.
- Descriptor: número entero asociado a un archivo abierto (conexión de red, un terminal, etc)

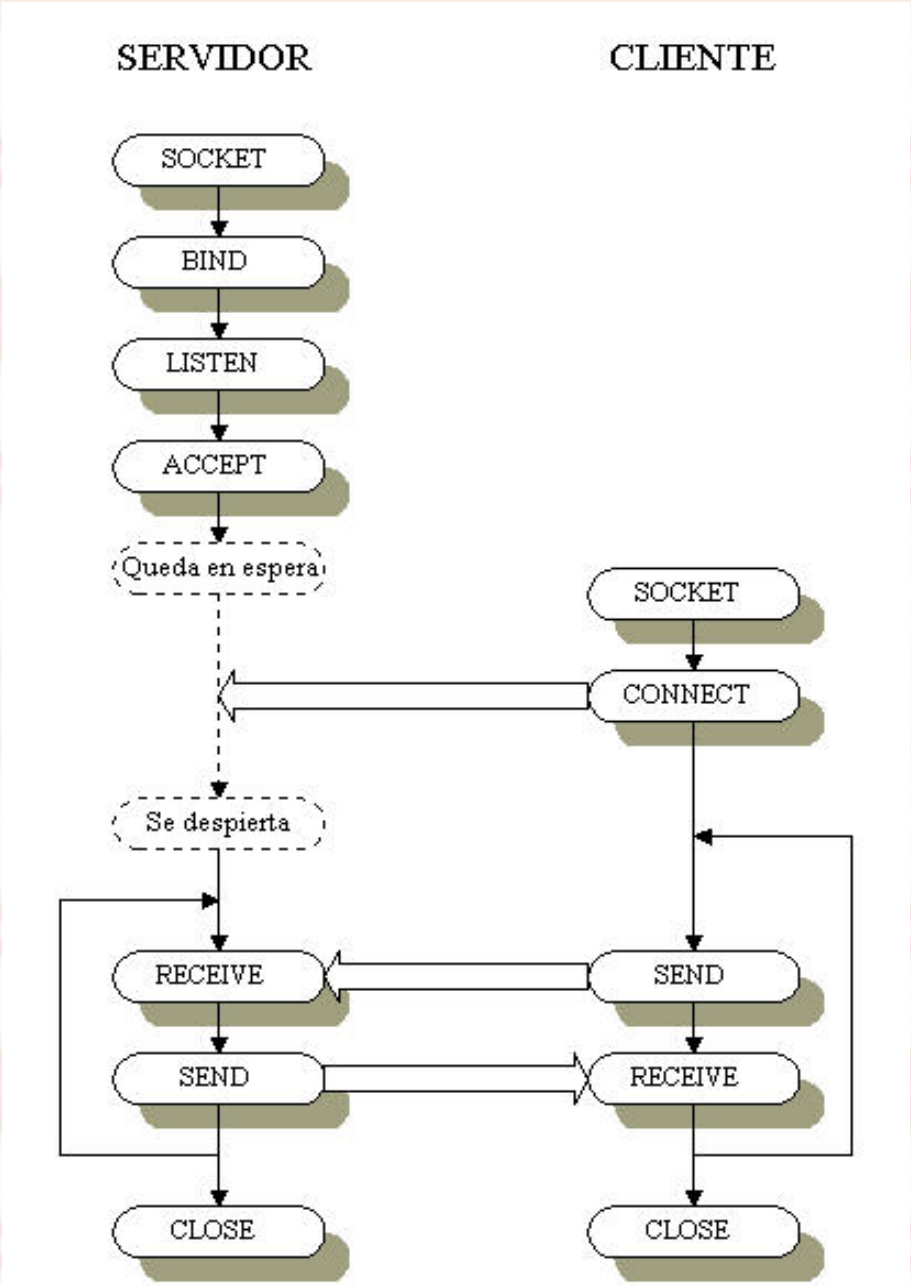
Sockets TCP y UDP

- Los sockets se crean dentro de un dominio de comunicación.
- Si se encuentran dentro del mismo sistema el dominio se llama AF_UNIX
- Si se encuentran en distintos sistemas y se encuentran unidos mediante TCP/IP el dominio se llama AF_INET.
- Tipos de Sockets en el dominio AF_INET:
  - Socket Stream
  - Socket Datagram
  - Socket Raw



Sockets TCP y UDP

- Sockets de Flujo (Sock_Stream): están libres de errores, utilizan TCP y garantizan el orden de los objetos durante la transmisión.
- Sockets de Datagramas (Sock_Dgram): Usan UDP, no necesitan de una conexión accesible, se construye un paquete de datos con información de su destino y se envía afuera sin necesidad de una conexión.
- Sockets Raw: Son empleados para desarrollar nuevos protocolos de comunicación o para hacer uso de facilidades ocultas de un protocolo existente.


SERVIDOR
CLIENTE
SOCKET
BIND
LISTEN
ACCEPT
Queda en espera
SOCKET
CONNECT
Se despierta
RECEIVE
SEND
SEND
RECEIVE
CLOSE
CLOSE

Byte Order

- Network byte order y Host byte order : dos formas en las que el sistema puede almacenar los datos(bytes) en memoria.
- Si al almacenar un short int (2 bytes) o un long int (4 bytes) en RAM, en la posición más alta se almacena el byte menos significativo = Network byte order

Byte Order

- Cuando se envía y recibe datos, se aplican la siguientes reglas:
  - Todos los bytes que se transmiten hacia la red, sean números IP o datos, deben estar en network byte order.
  - Todos los datos que se reciben de la red, deben convertirse a host byte order.
  - Funciones de conversión en C:
    - htons() - host to network short - convierte un short int de host byte order a network byte order.
    - htonl() - host to network long - convierte un long int de host byte order a network byte order.
    - ntohs() - network to host short - convierte un short int de network byte order a host byte order.
    - ntohl() - network to host long - convierte un long int de network byte order a host byte order.
  - Ejemplo: port = htons (4449); // Convertimos a network byte order el número de port que utilizamos.

Creación de un socket

- Para que una máquina pueda comunicarse con otra debe de existir un socket entre ellas.
- Los socket se crean llamando a la función socket()
- **dssock = socket (int dominio, int tipo, int protocolo);**

- dssock Es el descriptor de socket devuelto. Se utiliza para conectarse, recibir conexiones, enviar y recibir datos, etc.
- dominio Dominio donde se realiza la conexión.
- tipo Podrá ser SOCK_STREAM o SOCK_DGRAM o SOCK_RAW.
- protocolo 0 (cero, selecciona el protocolo más apropiado).

Nombrando la conexión

- El socket se crea sin nombre, hay que darle un nombre para poder recibir conexiones.
- bind () se utiliza para darle un nombre al socket, es decir, una dirección IP y número de puerto del host local por donde escuchará.
- Es necesario llamar a bind() cuando se está programando un servidor.
- Cuando se está programando un cliente generalmente no se utiliza esta función, el kernel le asignará al socket la dirección IP y número de puerto disponible.

Identificando la dirección

- **int bind(int dssock, struct sockaddr *addr, int addrtam)**
- dssock: Descriptor de socket obtenido de la función socket().
- addr : Es un apuntador a una estuctura sockaddr que contiene la IP del host local y el num. de puerto que se va a asignar al socket.
- addrtam : Es ell tamaño de la estuctura sockaddr.
- Ejemplo :
- struct sockaddr_in sin;
- bind (sockfd, (struct sockaddr *) &sin, sizeof (sin));

Estructuras

- Almacenan el nombre del socket.
- struct sockaddr{
- unsigned short sa_family; // AF_*
- char sa_data[14]; // Direccion de protocolo.
- };
- struct sockaddr_in {
- short int sin_family; // AF_INET
- unsigned short sin_port; // Numero de puerto.
- struct in_addr sin_addr; // Dirección IP.
- unsigned char sin_zero[8]; // Relleno.
- };
- struct in_addr{
- unsigned long s_addr; // 4 bytes.
- };

Otras funciones

- **inet_addr()**
- Convierte una dirección IP (notación números y puntos), en un unsigned long, retorna la dirección en network byte order. Retorna -1 si hubo error.
- **listen()**
- Se llama desde el servidor, habilita el socket para que pueda recibir conexiones. Solo se aplica a sockets tipo SOCK_STREAM.
- **accept()**
- Se utiliza en el servidor, con un socket habilitado para recibir conexiones (listen()). La llamada a accept() no retornará hasta que se produce una conexión o es interrumpida por una señal.

Otras funciones

- **connect()**
- Inicia la conexión con el servidor remoto, lo utiliza el cliente para conectarse.
- **read() y write()**
- Después de establecer la conexión, se puede comenzar con la transferencia de datos. Estas dos funciones son para realizar transferencia de datos sobre sockets stream.
- **int gethostname (char *hostname, size_t size)**
- Retorna el nombre del sistema donde esta ejecutandose el programa.El nombre puede ser utilizado por gethostbyname() para determinar la direccion IP del host local.