1. ¿Qué es un puerto?

En el nivel de software, dentro de un sistema operativo, un puerto es una construcción lógica que identifica un proceso específico o un tipo de servicio de red. Los puertos se identifican para cada protocolo y combinación de direcciones mediante números sin asignar de 16 bits, comúnmente conocido como el número de puerto.

2. ¿Cómo están formados los endpoints?

Un endpoint es una combinación de una dirección IP y un número de puerto. Cada conexión TCP puede identificarse de forma única por sus dos endpoints. De esa manera puede tener múltiples conexiones entre su host y el servidor.

3. ¿Qué es un socket?

Un socket es endpoint de un enlace de comunicación de dos vías entre dos programas que se ejecutan en la red.

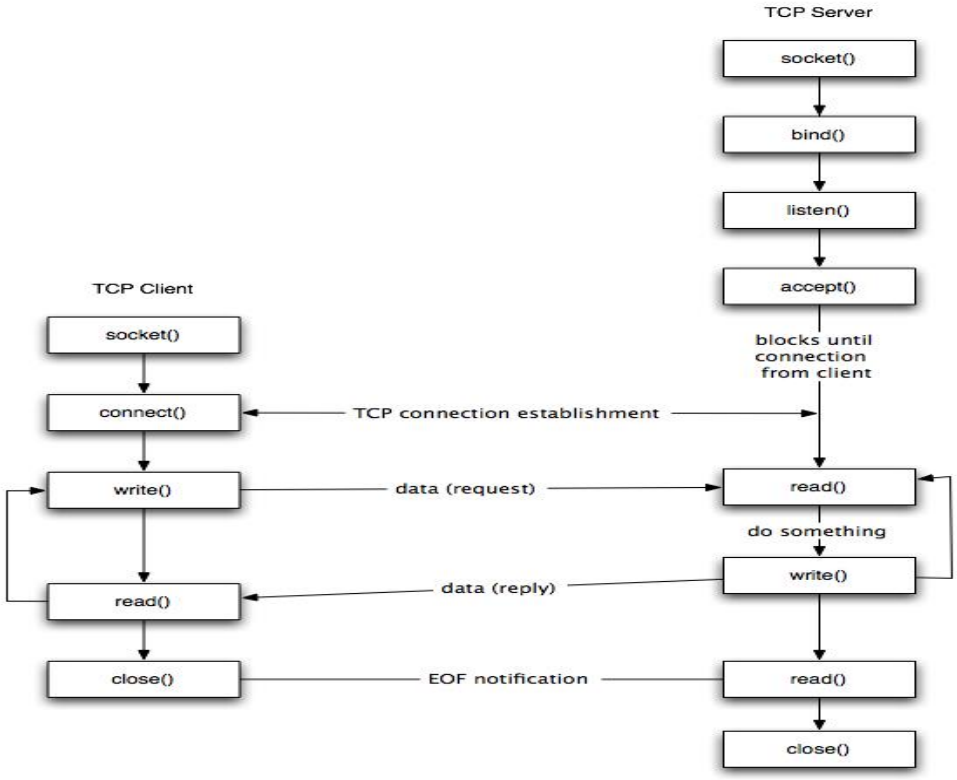
Un socket es similar a una toma de corriente eléctrica normal de la forma en que ambas se usan para hacer una conexión a otra ubicación. Más precisamente, un socket de red es algo que se abre o se cierra como un archivo, pero en lugar de leer o escribir datos en un disco, está enviando y / o recibiendo paquetes de red.

4. ¿A qué capa del modelo TPC/IP pertenecen los sockets? ¿Porque?

A la capa de transporte normalmente. Un socket está vinculado a un número de puerto para que la capa TCP pueda identificar la aplicación a la que están destinados los datos.

5. ¿Cómo funciona el modelo cliente-servidor con TCP/IP Sockets?

El proceso completo se puede dividir en los siguientes pasos:



El proceso completo se puede dividir en los siguientes pasos:

TCP Cliente –

- Crea un socket usando la función socket ();
- Conecte el zócalo a la dirección del servidor usando la función connect ();
- Envíe y reciba datos mediante las funciones de lectura () y escritura ().
- Cierre la conexión mediante la función close ().Vuelva al paso 3.

TCP Server –

• Crear un socket con la función socket ();

• Enlazar el socket a una dirección usando la función bind ();

• Escuche las conexiones con la función listen ();

• Aceptar una conexión con la llamada al sistema de la función accept (). Esta llamada generalmente se bloquea hasta que un cliente se conecta con el servidor.

• Enviar y recibir datos por medio de enviar () y recibir ().

• Cierre la conexión mediante la función close ().

6. ¿Cuáles son las causas comunes por la que la conexión entre cliente/servidor falle?

- El cliente informa: Connection refused. El servidor informa: nada.
- El cliente informa: Connection error: Connection reset by peer. El servidor informa: msg="Refused connection from ..." subroutine="libwrap".
- El cliente informa: Session key negotiation failed. El servidor informa: Invalid connection attempt: Session key mismatch
- El cliente informa: Session key negotiation failed. El servidor informa: Invalid connection attempt: Not in client list.
- El cliente informa: File download failed. El servidor informa: File not accessible.
- El cliente informa: Invalid connection state. El servidor informa: Invalid connection attempt: Signature mismatch.
- El servidor informa: Restart without prior exit for a client.

7. Diferencias entre sockets UDP y TCP

Hay algunas diferencias fundamentales entre TCP y UDP sockets. UDP es un protocolo de datagramas sin conexión, no confiable (TCP está orientado a la conexión, es confiable y se basa en la transmisión).

- La pila TCP se ocupa de los datos que se envían a la red y se envían al receptor, retransmitiéndolos hasta que el receptor los confirma.
- TCP también se encarga del control de flujo, es decir, transmite los datos a una velocidad adecuada para la conexión de red y el receptor.
- TCP garantiza que el receptor obtenga los datos exactamente una vez y en el orden correcto.
- UDP, el programador gestiona la transmisión a la red directamente, y tiene que hacerse cargo de los paquetes perdidos y fuera de orden, así como del control de flujo y la fragmentación de los datos a los paquetes que se pueden transmitir a través de la conexión de red.

8. Diferencia entre sync & async sockets?

La principal diferencia es que sincrónico implica bloquear un hilo que, de lo contrario, haría otras cosas útiles, o dedicar un hilo a cada conexión. De cualquier manera, esto no escala muy bien. Para aplicaciones simples con poca o solo una conexión activa, podría estar bien.

Pero para cualquier escenario en el que necesite manejar un número significativo de conexiones simultáneas, las API asíncronas son las únicas que proporcionan un rendimiento adecuado. Además, en cualquier escenario interactivo (es decir, cuando tiene que lidiar con la entrada y salida del usuario), el enfoque asíncrono se integra más fácilmente con la interfaz de usuario