1. ¿Qué es un puerto?

En informática, un puerto es una interfaz a través de la cual se pueden enviar y recibir los diferentes tipos de datos.

La interfaz puede ser de tipo física (hardware) o puede ser a nivel lógico o de [software](https://es.wikipedia.org/wiki/Software), en cuyo caso se usa frecuentemente el término puerto lógico (por ejemplo, los [puertos de redes](https://es.wikipedia.org/wiki/Puerto_de_red) que permiten la transmisión de datos entre diferentes computadoras).

Se denomina “puerto lógico” a una zona o localización de la [memoria de acceso aleatorio](https://es.wikipedia.org/wiki/Memoria_de_acceso_aleatorio) (RAM) de la computadora que se asocia con un puerto físico o un canal de comunicación, y que proporciona un espacio para el almacenamiento temporal de la información que se va a transferir entre la localización de memoria y el canal de comunicación.

1. ¿Qué es un socket?

Un EndPoint es un dispositivo informático remoto que se comunica con una red a la que está conectado. Los ejemplos de Enpoint incluyen:

-Ordenadores de escritorio  
-Portátiles  
-Móviles  
-Tablets  
-Servidores

-Estaciones de trabajo.

Un EndPoint está conformado por un número de IP y un número de Puerto.

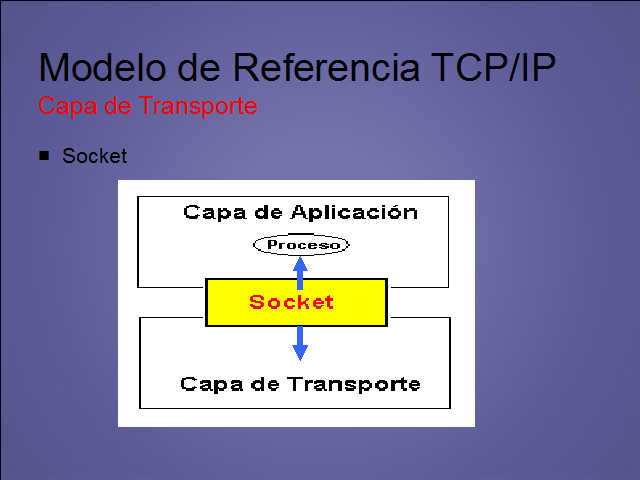
1. ¿Qué es un Socket?

Un socket es conocido como un tipo de software que actúa como un punto final que funciona estableciendo un enlace de comunicación de red bidireccional entre el extremo del servidor y el programa receptor del cliente. También se le conoce como un punto final en un canal de comunicación bidireccional. Estos sockets se realizan y movilizan junto con un conjunto de peticiones de programación que se identifican como llamadas de función, lo que técnicamente se denomina interfaz de programación de aplicaciones (API). Un socket es capaz de simplificar el funcionamiento de un programa porque los programadores ahora sólo tienen que preocuparse de manipular las funciones del socket y esto les permite confiar en el sistema operativo para transportar los mensajes a través de la red correctamente.

1. ¿A qué capa del modelo TCP/IP pertenecen los sockets? ¿Por qué?

La comunicación extremo a extremo entre aplicaciones (Procesos) se hace a través de un Soket Socket Interfaz de comunicación entre un proceso de aplicación y la capa de transporte Puerta que permite pasar los datos de la red a los procesos Los sockets pueden ser TCP o UDP Los sockets están asociados a puertos.

Un Socket se encuentra en medio de la capa de Transporte y Aplicación y es el nexo para que ambas se comuniquen.



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

Un Socket se rige por un flujo específico de eventos para que funcione. Para un modelo cliente-servidor orientado a la conexión, el socket en el proceso del servidor espera la petición de un cliente. Para hacer esto, el servidor necesita principalmente establecer una dirección que los clientes puedan usar para encontrar y conectarse al servidor. Cuando se establece una conexión con éxito, el servidor esperará a que los clientes soliciten un servicio. El intercambio de datos cliente-servidor tendrá lugar si el cliente se conecta al servidor a través del socket. El servidor responderá a la solicitud del cliente y enviará una respuesta.

La mayoría de las veces, las URL y sus conexiones se utilizan para acceder a Internet. Los programas requerirán un simple enlace de comunicación entre el cliente del programa y el servidor. Esta función está asociada a un socket que ayuda a vincular los extremos del cliente y del servidor del programa. Si un cliente comienza a establecer comunicación con el servidor, se realizará una conexión fiable entre el servidor y el cliente a través del canal de comunicación TCP. Con este tipo de comunicación, tanto el cliente como el servidor pueden leer o escribir en sockets que están conectados a un canal de comunicación específico.

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

* El servidor no se encuentra habilitado y por esto el cliente no puede conectarse al mismo.
* El puerto de recepción del servidor está bloqueado o cerrado.
* El cliente intenta utilizar un endpoint que se encuentra ya en uso por otro.
* El servidor intenta enviar datos a un cliente que ya se ha desconectado.
* El cliente utiliza una dirección de host y/o un puerto inválido para conectarse al servidor.

1. Diferencias entre un sockets UDP y TCP.

La principal diferencia entre ambos es que el UDP necesita que le entregemos paquetes de datos que el usuario debe construir, mientras el TCP admite bloques de datos (cuyo tamaño puede ir desde 1 bytes hasta muchos K bytes, dependiendo de la implementación) que serán empaquetados de forma transparente antes de ser transmitidos.

Existe además otra diferencia importante. Tanto los paquetes de datos UDP como los segmentos TCP (este es el nombre que reciben los paquetes TCP) pueden perderse (muy rara vez llegan al destino correcto con errores). Si un paquete se pierde el UDP no hace nada. Por el contrario, si un segmento se pierde el TCP lo retransmitirá, y este proceso durará hasta que el segmento ha sido correctamente entregado al host receptor, o se produzca un número máximo de retransmisiones.

Finalmente, en aplicaciones en tiempo real es necesario también tener en cuenta una cosa. En el UDP controlamos qué datos viajan en cada paquete. En el TCP esto no es posible porque el empaquetamiento es automático. De hecho, el TCP espera un tiempo prudencial a tener bastantes datos que transmitir antes de enviar un segmento ya que esto ahorra ancho de banda. Si es importante que los datos tarden el mínimo tiempo posible en llegar al receptor el UDP es la mejor opción. En este sentido se dice que el UDP tiene una menor latencia que el TCP.

1. Diferencias entre sync & async sockets.

Las diferencias es que uno es un proceso bloqueante y el otro no. La asincronía es esencial para las actividades que son potencialmente bloqueantes, por ejemplo cuando su aplicación accede a la web. El acceso a un recurso web a veces es lento o con retrasos. Si esta actividad está bloqueada dentro de un proceso sincrónico, la aplicación completa debe esperar. En cambio con un proceso asíncrono, la aplicación puede continuar con otro trabajo que no depende del recurso web hasta que la tarea potencialmente bloqueante o síncrona finaliza.