Ejercicio Teórico

Contesta las preguntas en azul y rellena los lugares dónde hay puntos.

1. Explica que es un socket y qué elementos básicos necesita.

   Un socket es uno de los extremos en la comunicación bidireccional entre dos programas que se ejecutan en una red y permite la transferencia de información entre ambos programas.

   Elementos básicos:
   - **Dirección IP**: Especifica la dirección del dispositivo en el que se encuentra el socket para que este pueda recibir o enviar información.
   - **Número de puerto**: Especifica el puerto del dispositivo identificado por la dirección IP desde el que el socket se comunicará. De este modo, dentro de una misma IP puede hacer varios sockets comunicándose.

2. Qué tipos de socket hay y para que se utilizan (nombra 4 tipos)

   Stream Sockets (sockets de flujo): siguen el protocolo TCP. Se usan para establecer conexiones orientadas a conexión, es decir, con control de errores y retransmisiones en caso que se pierdan datos por el camino. Se usan para aplicaciones en las que la fiabilidad de la transferencia de información sea primordial. Un ejemplo de estas aplicaciones son el correo electrónico o la transferencia de archivos.

   Datagram Sockets (sockets de datagrama): siguen el protocolo UDP. Se utilizan para comunicaciones no orientadas a conexión, es decir, los dispositivos no saben que se van a comunicar con alguien hasta que les llega un mensaje. Son más rápidos y eficientes (menor latencia) que los de tipo stream pero menos confiables. Este tipo de sockets se usa comúnmente para aplicaciones en tiempo real donde tiene mayor importancia la velocidad de transmisión de datos que la fiabilidad. Algunos ejemplos de este tipo de aplicaciones son el streaming de vídeo o los juegos en línea.

   Raw Sockets: pueden utilizar varios protocolos de bajo nivel, como ICMP. Permiten un mayor control sobre la comunicación a nivel de paquete, permitiendo la comunicación sin el procesamiento de la información que hacen habitualmente las capas superiores del modelo OSI. Se usan normalmente para tareas de diagnóstico o de análisis de redes, normalmente, requieren de privilegios de administrador.

   Web Sockets: siguen el protocolo WebSocket, que se ejecuta sobre TCP. Proporcionan una comunicación bidireccional de bajo retardo. Son la mejor opción para aplicaciones web en tiempo real. Crean una conexión persistente y permiten una transferencia de datos sin necesidad de una solicitud-respuesta como en HTTP. Se utilizan para aplicacione web interactivas en las que es necesaria la transmisión de datos en las dos direcciones a través del navegador web. Algunos ejemplos de estas aplicaciones son chats en línea, juegos multijugador…

3. Qué es una arquitectura Cliente – Servidor

   Es un modelo de diseño utilizado en informáticas y redes en el que un sistema se divide en dos componentes principales: el cliente y el servidor.

   Cliente: Este componente del sistema solicita servicios o recursos al servidor. Son aplicaciones o dispositivos que interactúan con el usuario final, recopilan sus solicitues y las envían al servidor.

   Servidor: Este componente proporciona los servicios solicitados por el cliente. Escucha las solicitudes entrantes, las procesa y responde en función de estas. Pueden ser máquinas físicas o software que se ejecute en máquinas dedicadas.

4. Pon un ejemplo detallando los pasos de una comunicación Cliente-Servidor con Stream Socket (orientado a conexión). Para cada función indica que hace y qué parámetros necesita.

Ejemplo (ampliando cada punto indicado):

- **El servidor se prepara:**

1. El servidor crea un socket con el método **socket(socket.AF_INET, socket.SOCK_STREAM)**
2. Luego vincula el socket a una dirección **bind(hostname, número de puerto)**
3. A continuación el servidor ejecuta el método **listen(<número máximo de conexiones que acepta el socket al mismo tiempo>)** para escucha las solicitudes de conexión de los cliente que quieran sus servicios.
4. El servidor recibe una solicitud de un cliente y ejecuta un **accept()** para aceptarlo. Se hace el TCP 3-way handshake.
5. Intercambia información con el cliente leyendo los mensajes recibidos mediante la función **recieve(bufsize)** o **recv(bufsize)** y escribiendo las respuestas mediante las funciones **send(mensaje)** y **sendall(mensaje)** los que envía al cliente.
6. El cliente notifica el cierre de la conexión y el servidor cierra su extremo de la conexión con la función **close()**.

- **El cliente se prepara:**

1. El cliente crea un socket con el método **socket(socket.AF_INET, socket.SOCK_STREAM)**
2. El cliente ejecuta el método **connect(hostname, número de puerto)** indicando el hostname y el número de puerto del servidor al que quiere solicitar una información o un servicio.
3. El cliente recibe el accept() del servidor y se hace el 3-way hadhsake. La conexión entre el cliente y el servidor queda establecida. Entonces envía su solicitud mediante la función **send(mensaje)** o **sendall(mensaje)**
4. El cliente recibe la respuesta del servidor mediante la función **recv(bufsize)**
5. El cliente cierra la conexión y envía la notificación de cierre de conexión al servidor con la función **close()**.

- **Comunicación entre el servidor y el cliente:**

1. El servidor ha recibido la solicitud del cliente y ejecuta un Accept() para vincularse con el cliente. ¿Cómo lo hace? Accept() devuelve una tupla (cliente, dirección). Cliente es un nuevo socket creado en el servidor para atender las peticiones del cliente.
2. El cliente envía una solicitud al servidor con el método **send(mensaje)**
3. El servidor recibe la solicitud en el socket cliente con el método **receive(bufsize)**
4. Ahora el servidor envía la respuesta a la solicitud con el método **send(mensaje)** o **sendall(mensaje)**
5. El cliente recibe la respuesta del servidor con el método **recv(bufsize)**
6. Los pasos 2, 3, 4 y 5 se repiten tantas veces como solicitudes tenga el cliente.
7. Cuando el cliente no tiene más solicitudes cierra la conexión con la función **close()**, lo que cierra el socket del cliente y envía una señal al servidor.
8. El servidor recibe la señal de cierre del socket del cliente.
9. El servidor cierra su socket con la función **close()**.

5. Pon un ejemplo detallando los pasos de una comunicación Cliente-Servidor con Datagram Socket (no orientado a conexión). Para cada función indica que hace y qué parámetros necesita.

- **El servidor se prepara:**

1. El cliente crea un socket con el método **socket(socket.AF_INET, socket.SOCK_DGRAM)**
2. El servidor vincula su socket a una dirección con la función **bind(hostname, número de puerto)**
3. El servidor recibe la solicitud de un cliente con la función **recvfrom(bufsize)**. Esta función devuelve una tupla (mensaje, addr), donde el mensaje es la solicitud del cliente y addr es la dirección de este. Se guarda la dirección del cliente para poder devolverle información.
4. El servidor responde al cliente que le ha hecho la solicitud mediante la función **sendto(mensaje, addr)**.
5. Cuando termina la función del servidor ya no espera ningún cliente más o con alguna condición de finalización se usa la función **close()** para cerrar el servidor.

- **El cliente se prepara:**

1. El cliente crea un socket con el método **socket(socket.AF_INET, socket.SOCK_DGRAM)**
2. El cliente envía una solicitud al servidor con la función **sendto(mensaje, addr)**, donde el mensaje es la solicitud del cliente y addr es la dirección del servidor al que se va a preguntar representada por una tupla (ip, número de puerto).
3. El cliente recibe la respuesta del servidor con la función **recvfrom()**, que devuelve una tupla (mensaje, addr) donde mensaje es la respuesta del servidor y addr es la dirección del servidor.
4. Cuando el cliente no quiere hacer más cosultas al servidor utiliza la función **close()** para cerrar su socket.

- **Comunicación entre el servidor y el cliente:**

1. El servidor, con su socket de tipo dgram creado y asociado a una dirección, utiliza la función **recvfrom()** y se queda a la espera de que llegue alguna solicitud de algún cliente.
2. El cliente, con su socket de tipo dgram creado envía una solicitud al servidor con la función **sendto(mensaje, addr)**, donde mensaje es la solicitud y addr es una tupla (dirección, número de puerto) que indica a quién se envía el mensaje.
3. El servidor, que estaba esperando alguna solicitud la recibe con la función **recvfrom(bufsize)** que proporciona una tupla (mensaje, addr) donde mensaje es la solicitud del cliente y addr es una tupla (dirección, número de puerto). Se guarda la dirección del cliente para poder responderle (necesario porque, con los socket dgram no hay una conexión establecida y, por tanto, si no guardase la dirección no sabría a quién responder).
4. El servidor procesa la solicitud del cliente y le responde con la función **sendto(mensaje, addr)**, donde mensaje es la respuesta y addr es una tupla (ip, número de puerto) que representa la dirección del cliente al que se le responde.
5. El cliente recibe la respuesta del servidor con la función **recvfrom(bufsize)**, que devuelve una tupla (mensaje, addr) donde mensaje es la respuesta del servidor y addr es una tupla (dirección, número de puerto) que indica la dirección del servidor.
6. Si el cliente no quiere hacer más consultas al servidor cierra su socket con la función **close()**.
7. El cliente seguiría esperando nuevas solicitudes o, en caso de que ya no espere más o cumpla alguna condición de cierre, cerraría su socket con la función **close()**.

6. Enumera las diferencias entre una comunicación Cliente-Servidor con Stream Socket y Datagram Socket.

1- **Conexión**: En la comunicación mediante Stream Socket se establece una conexión antes de la transferencia de datos y proporciona un flujo de datos bidireccional y fiable. En la comunicación entre sockets de tipo Datagram Socket los mensajes se envían como datagramas independientes sin estableces una conexión previa. Se podría decir que utiliza una conexión sin conexión. El

servidor estaría esperando sin más a que le llegue una solicitud en forma de mensaje y dirección y respondería a esa dirección sin tener un “canal” abierto directamente con ella.

2- **Fiabilidad**: El Stream Socket proporciona una transferencia de datos fiable. Los datos se entregan en el orden correcto y hay mecanismos de control y retransmisión en caso de errores. No es el caso de los Datagram Socket, que no garantizan la fiabilidad de los datos recibidos. Los datagramas pueden perderse o llegar en un orden diferente al orden en el que se enviaron.
3- **Flujo de datos**: Los Stream Sockets ofrencen un flujo de datos continuo y bidireccional. Los datos se envían como un flujo de bytes y se reciben de la misma manera. Los Datagram Sockets envían bloques discretos de datos llamados datagramas. Cada datagrama es independiente y puede llegar en cualquier orden.
4- **Latencia**: La comunicación mediante Stream Sockets, normalmente, tiene una latencia menor, dado que la conexión que se establece permite una transferencia sin errores y las medidas de control implican un aumento del tiempo de transferencia. Los Datagram Sockets consiguen una menor latencia, ya que cada datagrama se manera de forma independiente y no hay mecanismos de control.
5- **Uso de memoria**: Los Stream Sockets pueden requerir mayor cantidad de memoria debido a la gestión de la conexión y la necesidad de mantener un flujo continuo de datos. Los Datagram Sockets son más eficientes en este campo, ya que cada datagrama es independiente y no se necesita mantener una conexión persistente.