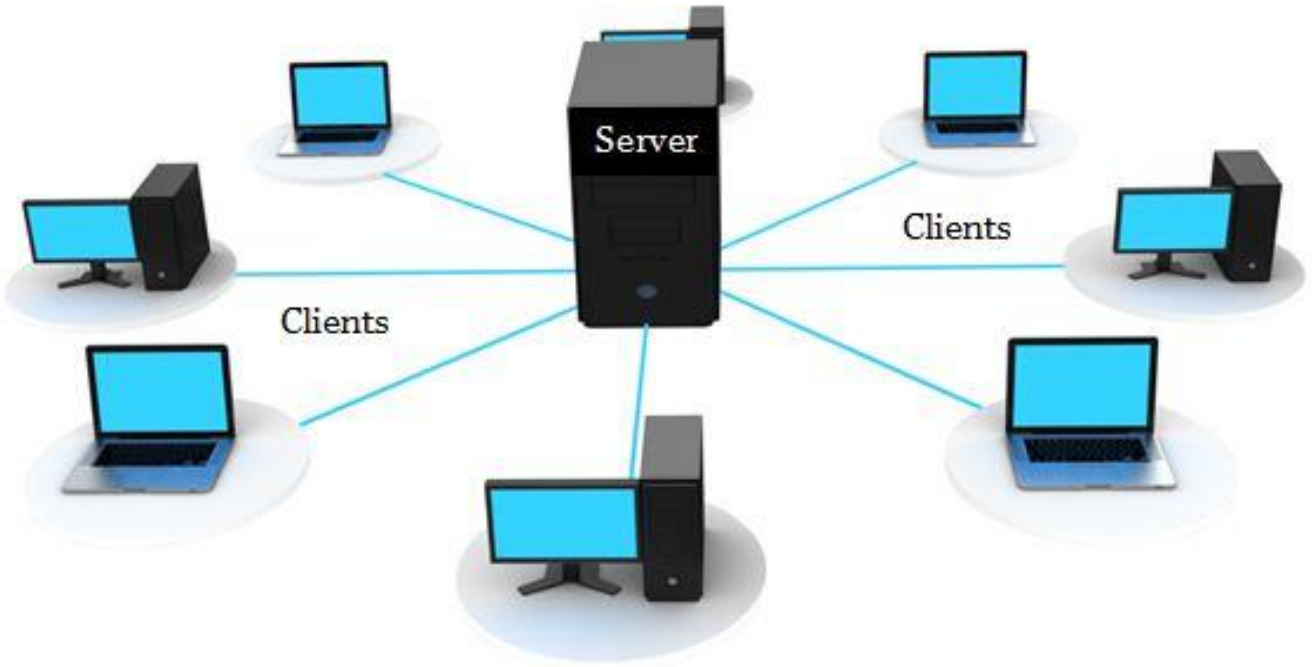
MultiXat

Documentación



Mariona Farré Tapias - 26890730G
Marc Pérez Guerrero - 48076170Z
Asignatura: XACO
Curso: 2022/ 2023

Documentación del código

Estructuras básicas

El archivo **ServerStructs.py** contiene las estructuras básicas utilizadas en el servidor. Estas estructuras son las siguientes:

SERVER

- Atributos:
    - **server_addr:** La dirección IP del servidor.
    - **server_port**: El puerto de comunicación del servidor.
    - **server_socket**: El socket del servidor utilizado para aceptar conexiones de clientes.
    - **server_clients**: Una lista que almacena los clientes conectados al servidor.
    - **server_canals**: Una lista que almacena los canales disponibles en el servidor.

CANAL

- Atributos:
    - **canal_name**: El nombre del canal.
    - **canal_clients**: Una lista que almacena los clientes que están conectados al canal.

CLIENTE

- Atributos:
    - **client_name**: El nombre del cliente.
    - **client_connection**: El socket de conexión del cliente.
    - **client_canal**: El canal al que pertenece el cliente.

Servidor

El archivo **TCPServer.py** contiene las funciones del servidor. Son las siguientes:

server_start()

Esta función es responsable de iniciar el servidor. Realiza los siguientes pasos:

1. Vincula el socket del servidor a la dirección IP y al puerto especificados.
2. Comienza a escuchar en el puerto inicializado.
3. Añade el canal principal a la lista de canales del servidor.
4. Inicia un bucle infinito para aceptar conexiones de clientes.
5. Cada vez que se establece una conexión con un cliente:
    - Crea una nueva instancia de la estructura CLIENTE.
    - Guarda el nombre del cliente recibido a través del socket de conexión.
    - Establece la conexión del cliente.
    - Asigna al cliente el canal principal como canal actual.
    - Añade al cliente a la lista de clientes del servidor.
    - Añade al cliente al canal principal.
    - Imprime un mensaje de notificación de la conexión del cliente.
    - Inicia un hilo de ejecución para manejar los eventos del cliente.

manage_client(client)

Esta función es responsable de gestionar las interacciones con un cliente específico. Realiza los siguientes pasos:

1. En un bucle infinito, espera recibir paquetes de datos del cliente.
2. Si se recibe un paquete de datos vacío, se sale del bucle y se finaliza la función.
3. Decodifica el mensaje recibido.
4. Prepara el mensaje para su procesamiento.
5. Verifica si el mensaje es una instrucción especial llamando a la función **manage_instructuion()**.
6. Si es una instrucción especial:
    - Imprime el mensaje junto con el nombre del cliente que lo envió.
    - Envía el mensaje a todos los clientes en el canal actual del cliente (excepto al cliente emisor) llamando a la función **send_message()**.
7. Si no es una instrucción especial, se ignora el mensaje.

send_message(message, from_client)

Esta función se encarga de difundir un mensaje a todos los clientes en el canal actual del cliente emisor. Realiza los siguientes pasos:

1. Recorre todos los clientes del canal actual del cliente emisor.
2. Verifica que el cliente no sea el emisor del mensaje.
3. Envía el mensaje al cliente.

get_canal_index(canalName)

Esta función devuelve el índice del canal en la lista de canales del servidor que coincide con el nombre especificado. Si no se encuentra el canal, devuelve **-1**.

get_client_index(clientName)

Esta función devuelve el índice del cliente en la lista de clientes del servidor que coincide con el nombre especificado. Si no se encuentra el cliente, devuelve **-1**.

manage_instructuion(message, client)

Esta función maneja las instrucciones especiales enviadas por los clientes. Realiza las siguientes acciones en función del tipo de instrucción recibida:

- "**CREA**": Crea un nuevo canal con el nombre especificado si no existe previamente.
- "**CONFIDENCIAL**": Crea un nuevo canal confidencial con el nombre especificado si no existe previamente y una contraseña.
- "**ELIMINAR**": Elimina un canal con el nombre especificado si existe.
- "**ENTRA**": Entra en el canal con el nombre especificado si existe y sale del anterior.
- "**NOU_NOM_CANAL**": Cambia el nombre de un canal existente por el nuevo nombre especificado.
- "**PRIVAT**": Envía un mensaje privado al cliente especificado.
- "**ON_ESTIC**": Muestra el canal en el que está el cliente que lo ha pedido.
- "**MOSTRA_CANALS**": Muestra una lista de los nombres de todos los canales disponibles en el servidor.
- "**MOSTRA_USUARIS**": Muestra una lista de los nombres de los clientes en el canal actual del cliente emisor.
- "**MOSTRA_TOTS**": Muestra una lista de los nombres de todos los clientes conectados al servidor.
- "**HELP**": Envía al cliente una lista de las opciones disponibles.
- "**SORTIR**": Realiza una acción de salida específica (no implementada en el código).

Cliente

TCPClient.py, establece la comunicación con el servidor TCP y permite a los usuarios enviar y recibir mensajes en el chat.

1. **Variables de configuración:**
    - **serverName**: La dirección IP del servidor al que se va a conectar el cliente. Actualmente, está configurado para *'localhost'*, lo que significa que se conectará al servidor en la misma máquina.
    - **serverPort**: El puerto de comunicación del servidor. Actualmente, está configurado para *12000*.
2. **Funciones auxiliares:**
    - **nombreUsuario()**: Solicita al usuario que introduzca su nombre y lo devuelve.
3. **Función principal cliente():**
    - Crea un socket de cliente TCP utilizando **socket(AF_INET, SOCK_STREAM).**
    - Obtiene el nombre de usuario utilizando la función nombreUsuario().
    - Establece la conexión TCP con el servidor utilizando **connect((serverName, serverPort)).**
    - Envía el nombre de usuario al servidor utilizando **send(UserName.encode())**.
    - Inicia un hilo de ejecución utilizando threading.Thread() y la función **recibir_mensajes()** para recibir y procesar los mensajes del servidor.
    - En un bucle infinito, espera que el usuario ingrese un mensaje y luego lo envía al servidor utilizando **send(chat.encode())**.
4. **Función recibir_mensajes(cSocket):**
    - En un bucle infinito, espera recibir mensajes del servidor utilizando **recv(1024)**.
    - Decodifica el mensaje recibido.
    - Imprime el mensaje en la consola del cliente en el formato **"EMISOR: MENSAJE"**.

Manual de ejecución

Guía de uso para ejecutar los códigos en Linux:

Paso 1: Preparación

1. Asegúrate de tener instalado Python en tu sistema Linux. Puedes verificar si Python está instalado ejecutando el siguiente comando en la terminal:
   python --version

2. Si Python no está instalado, puedes instalarlo utilizando el administrador de paquetes de tu distribución Linux. Por ejemplo, en Ubuntu puedes ejecutar: arduino
   sudo apt-get install python3

3. Guarda los archivos de código proporcionados en una ubicación conveniente en tu sistema.

Paso 2: Configuración del servidor

1. Abre una terminal y navega hasta la ubicación donde guardaste el archivo **TCPServer.py**.

2. Ejecuta el siguiente comando para iniciar el servidor:
   python3 TCPServer.py

3. Esto iniciará el servidor en la dirección IP predeterminada (*0.0.0.0*) y el puerto predeterminado (*12000*).

Paso 3: Configuración del cliente

1. Abre otra terminal y navega hasta la ubicación donde guardaste el archivo TCPClient.py.
2. Ejecuta el siguiente comando para iniciar el cliente:
   python3 TCPClient.py

1. El cliente solicitará que introduzcas tu nombre. Ingresa un nombre y presiona Enter.
2. Ahora puedes enviar y recibir mensajes en el chat. Escribe tus mensajes en la terminal del cliente y presiona Enter para enviarlos.
    a. Nota: El cliente y el servidor deben estar en la misma red y tener acceso a la dirección IP del servidor para comunicarse correctamente.
    b. Nota 2: Para ver las opciones disponibles, escribe HELP en la terminal.