Persistencia y datos transaccionales

Entrega semana 3

Subgrupo 8

Integrantes:

Andrés Torres Cadena

Nicolas Fernando Casallas Suarez

John Richard Merchán Martínez

Cristian Camilo Sabogal Lopez

Diego Roa

Docente

Politécnico Grancolombiano

2024

CONTENIDO

Introducción

El siguiente trabajo tiene como objetivo el desarrollo de un chat bidireccional utilizando sockets, para este trabajo, se utilizó el lenguaje de programación Python. De acuerdo con lo solicitado para este desarrollo, el sistema se compone por un servidor y un cliente, el servidor puede albergar la conexión de múltiples clientes los cuales podrán intercambiar mensajes de forma instantánea, esta comunicación entre si se da a través de un canal de red. De momento, y para esta entrega se brinda un desarrollo eficaz que cumple con lo requerido, una interfaz sencilla por línea de comandos y un chat bidireccional con un servidor que proporciona los sockets para este objetivo.

Objetivos

Objetivo general

Utilizando la tecnología de sockets, crear un sistema de chat bidireccional que permita la comunicación instantánea y en tiempo real entre dos o más clientes, utilizando el lenguaje de programación Python.

Objetivos específicos

1. Configurar el servidor mediante Sockets
    a. Crear un servidor mediante Python con la capacidad de aceptar múltiples conexiones de clientes.
    b. Crear la configuración de los clientes y que estos a su vez se conecten con el servidor y así iniciar la comunicación bidireccional.
2. Configuración de la interfaz de comunicación.
    a. Crear una interfaz gráfica o por la línea de comandos donde se pueda realizar la comunicación.
    b. Crear un método de salida para finalizar el chat

1. Fundamentos de programación con sockets

Los sockets son interfaces que permiten la comunicación entre nodos de una misma red, su uso es redes informáticas con el fin de permitir la comunicación entre procesos, ya sea en uno o varios servidores pertenecientes a una misma red. Esta comunicación presenta dos tipos principales:

1. Stream Socket: Conocido como socket de flujo, es utilizado para la comunicación en conexiones mediante protocolo TCP/IP, esta conexión se realiza con el fin de garantizar la entrega de la información en el mismo orden de salida. Para garantizar lo anterior, las conexiones se realizan y se verifican antes de iniciar la transmisión de la información.
2. Datagram Socket: Estos se encuentran configurados con el protocolo UDP (USER DATAGRAM PROTOCOL) y su uso frecuente para las comunicaciones sin conexión, esto significa que los datos son enviados de forma independiente la entrega completa y el orden de estos no es garantizada, esto quiere decir que como es sin conexión parte de la información puede quedar incompleta.

2. Ciclo de vida de un socket TCP

El ciclo de vida de un socket se define por la creación de este por el servidor, hasta el punto de finalización, los siguientes son cada uno de los aspectos de este ciclo:

1. Configuración en el servidor:
    a. Creación: En el servidor se crea el socket el cual recibe o escucha (LISTEN) las conexiones que entran
    b. Vinculación (BIND): En el servidor se vincula un puerto y una dirección IP específicos al socket.

c. Escucha (LISTEN): En el servidor, se activa el modo LISTEN del socket para que quede listo para recibir las conexiones.
d. Aceptación: Al estar en modo LISTEN el socket, recibe las conexiones entrantes y crea un nuevo socket para la comunicación con el cliente.
e. Comunicación: Intercambio de datos entre el cliente y el servidor.
f. Close: Al finalizar la comunicación entre ambos lados, cada uno cierra los sockets y la comunicación termina.

2. Configuración Cliente
   a. Creación: Se crea un socket en el equipo cliente
   b. Conexión: El cliente realiza la conexión al mediante la IP y el puerto del servidor
   c. Comunicación: Una vez establecida la conexión, servidor y cliente realizan el intercambio de información.
   d. Close: El equipo cliente al finalizar la comunicación cierra el socket

3. Estado del arte

Un socket es un concepto fundamental en la comunicación de red en la programación de computadoras. Representa un punto final de una conexión de red entre dos dispositivos, como computadoras, servidores, o incluso procesos dentro de una misma máquina.

El estado del arte en programación de sockets ha evolucionado junto con el avance de las tecnologías de red y los sistemas operativos. Algunos puntos destacados incluyen:

1. Estándares y Protocolos: Los sockets son una implementación de interfaces de programación de aplicaciones (APIs) definidas por estándares. En el caso de Internet los protocolos subyacentes son TCP (Transmission Control Protocol) y UDP (User Datagram Protocol).

2. Abstracciones y Bibliotecas: Los lenguajes de programación modernos suelen proporcionar bibliotecas y abstracciones de alto nivel para la manipulación de sockets. Estas simplifican la creación y gestión de conexiones de red, permitiendo a los desarrolladores concentrarse en la lógica de la aplicación.
3. Concurrencia y Asincronía: En aplicaciones de red modernas, la concurrencia y la asincronía son aspectos cruciales. Los modelos de programación asíncrona permiten manejar múltiples conexiones simultáneamente sin bloquear el hilo principal de ejecución.
4. Seguridad y Criptografía: En un mundo cada vez más preocupado por la seguridad, la programación de sockets también ha evolucionado para incluir mecanismos de seguridad como SSL/TLS para la encriptación de la comunicación.
5. Frameworks y Bibliotecas de Alto Nivel: Además de las API estándar proporcionadas por los sistemas operativos, existen frameworks y bibliotecas de alto nivel que simplifican aún más la programación de aplicaciones de red.
6. Evolución de los Paradigmas de Comunicación: Junto con los sockets tradicionales basados en TCP/IP, ha habido un aumento en el interés y el uso de tecnologías como WebSockets y gRPC, que proporcionan formas más eficientes y flexibles de comunicación en tiempo real y de alto rendimiento.

En resumen, el estado del arte en programación de sockets implica una combinación de estándares bien establecidos, bibliotecas de alto nivel y técnicas modernas de programación asincrónica para crear aplicaciones de red eficientes y seguras.

4. Diseño del modelo de comunicación por Sockets

De acuerdo con lo solicitado para este proyecto, la siguiente será la configuración de puerto y dirección IP:

SERVER IP: 172.0.0.1 (localhost)

Puerto: 59420

El servidor iniciará en la dirección IP 127.0.0.1 con el puerto 59420, para realizar conexión, los clientes se deberán conectar a ese puerto y dirección.

Chat Bidireccional

Para realizar esta conexión, es necesario crear dos programas uno para el servidor y otro para el cliente, el primero debe permitir las conexiones de varios clientes y este a su vez transmite los mensajes a los clientes que se encuentran conectados.

5. Sockets construidos para el taller

A continuación, se realiza un resumen detallado de las líneas de código Python para la creación de los sockets:

```
1    import socket
2    import threading
```

En la imagen se evidencia la importación de los módulos “Socket” y “threading” requeridos para las conexiones de red y también permitir las conexiones simultaneas, respectivamente.

```python
# Configuración del servidor
HOST = '127.0.0.1'
PORT = 59420
```

Definición de las variables tanto del servidor como del puerto, esta información es de acuerdo con lo requerido en el módulo.

```python
# Comunicación con el cliente
def manejar_cliente(cliente_socket, direccion):
    try:
        nombre_usuario = cliente_socket.recv(1024).decode('utf-8').split(':', 1)[1]
        clientes[cliente_socket] = nombre_usuario
        print(f"{nombre_usuario} se ha conectado desde {direccion}")
        actualizar_clientes()

        while True:
            mensaje = cliente_socket.recv(1024).decode('utf-8')
            tipo, contenido = mensaje.split(':', 1)
            if tipo == "MESSAGE":
                if contenido.strip().lower() == "chao":
                    print(f"{nombre_usuario} abandonó la conversación")
                    cliente_socket.send("DISCONNECT:".encode('utf-8'))
                    cliente_socket.close()
                    del clientes[cliente_socket]
                    actualizar_clientes()
                    break
                enviar_mensaje(nombre_usuario, contenido.strip())
            elif tipo == "DISCONNECT":
                print(f"{nombre_usuario} se ha desconectado")
                cliente_socket.send("DISCONNECT:".encode('utf-8'))
                cliente_socket.close()
                del clientes[cliente_socket]
                actualizar_clientes()
                break
    except:
        cliente_socket.close()
        del clientes[cliente_socket]
        actualizar_clientes()
```

Se define la función **manejar cliente**, la cual se encarga de gestionar la comunicación entre el servidor y un cliente en específico. Este parámetro toma dos configuraciones, la primera es **Cliente_Socket** esta configuración representa el socket del cliente conectado. La segunda configuración es **dirección**, la cual determina la dirección IP y el puerto del cliente.

```
# Iniciar el servidor
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
servidor.bind((HOST, PORT))
servidor.listen()
print("Servidor iniciado y esperando conexiones...")

while True:
    cliente_socket, direccion = servidor.accept()
    hilo = threading.Thread(target=manejar_cliente, args=(cliente_socket, direccion))
    hilo.start()
```

Por último, se crea el socket del servidor con el parámetro **socket.socket()**, el cual es vinculado a la dirección y puerto que ya se encuentran definidos con el parámetro **bind()**, posteriormente se procede a dejar en estado de escucha activa “listen()”. Se programa el bucle **while True** con el cual se deja en espera activa nuevas conexiones y para ello se deja el parámetro **acept()** para finalizar, al establecer una nueva conexión se parametriza el comando **threading.Thread()** el cual ejecuta la función **manejar_Cliente** encargado de gestionar la comunicación recién establecida con el cliente.

6. Diseño de red utilizado para el desarrollo

A continuación, se describe el diseño de la red que se utilizó para el desarrollo del trabajo el diseño básico, pero detalla cada uno de los parámetros utilizados para lograr el objetivo:

1. Cliente:
    a. Cada uno de los clientes que requiera unirse a la conversación, deberá conectarse con el servidor utilizando la dirección IP y el puerto definido.
    b. Los mensajes que envíen los clientes serán administrados por el servidor, el cual redireccionará el mensaje al chat correspondiente.
2. Servidor:
    a. El servidor recibe las conexiones entrantes en el puerto definido

   b. Mediante los subprocesos, el servidor gestiona las conexiones de los clientes de manera simultanea
   c. Recibe y transmite mensajes entre clientes.
3. Cliente y servidor:
   a. Con la creación de los sockets TCP/IP se establece la conexión entre cliente y servidor.
   b. El servidor administra los mensajes entre los clientes y los distribuye mediante los sockets.
4. Trafico
   a. En la red, permite la transferencia de datos entre el cliente y el servidor mediante la conexión TCP/IP que ha establecido entre los dispositivos.
   b. Los mensajes, datos, solicitudes de conexión y desconexión son los incluidos y permitidos por el servidor.
5. Sockets de dominio
   a. AF_NET (IPv4): Para comunicaciones IPv4
   b. SOCK_STREAM (TCP): Flujo de bytes ordenado y confiable.
   c. Protocolo IPPROTO_TCP Para usar TCP
6. Implementamos la lógica de la comunicación:
   a. Servidor:
      i. Crea un socket de escucha.
      ii. Espera conexiones entrantes.
      iii. Acepta la conexión del cliente.
      iv. Recibe y procesa datos del cliente.
      v. Envía una respuesta al cliente.
      vi. Cierra la conexión.

b. Cliente:

i. Crea un socket de conexión.

ii. Se conecta al servidor.

iii. Envía datos al servidor.

iv. Recibe y procesa datos del servidor.

v. Cierra la conexión.

7. Cliente:

```python
# Configuración del cliente
HOST = '127.0.0.1'
PORT = 59420
nombre_usuario = input("Nombre de usuario: ")

# Recibir mensajes del servidor
def recibir_mensajes():
    while True:
        try:
            mensaje = cliente.recv(1024).decode('utf-8')
            tipo, contenido = mensaje.split(':', 1)
            if tipo == "USER_LIST":
                print(f"Usuarios conectados: {contenido}")
            elif tipo == "MESSAGE":
                print(contenido)
            elif tipo == "DISCONNECT":
                print("Desconectado del servidor")
                cliente.close()
                break
        except:
            print("Desconectado del servidor")
            cliente.close()
            break

# Conexión al servidor
cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
cliente.connect((HOST, PORT))
cliente.send(f"CONNECT:{nombre_usuario}".encode('utf-8'))

# Iniciar hilo para recibir mensajes
hilo_recibir = threading.Thread(target=recibir_mensajes)
hilo_recibir.start()
```

8. Servidor:

```python
# Configuración del servidor
HOST = '127.0.0.1'
PORT = 59420
```

```python
# Iniciar el servidor
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
servidor.bind((HOST, PORT))
servidor.listen()
print("Servidor iniciado y esperando conexiones...")
```

7. Enlace del repositorio del código

https://github.com/AndresTorresCode/chatbid/tree/main/env

8. Enlace del video en YouTube

https://www.youtube.com/watch?v=3TLXAar_Cb4

9. Conclusiones

Con el desarrollo de esta actividad se aprendieron los términos de chat bidireccional, la configuración de sockets en Python, esto enriquece el conocimiento y prepara el terreno para nuevos desafíos en pro de la adquisición de fundamentos para la programación. Este proyecto, muestra de manera funcional el sistema que permite la comunicación entre varios clientes en tiempo real y que son utilizados en la actualidad por múltiples aplicaciones y redes sociales, además, muestra la eficiencia en la red la cual es la que permite la comunicación mediante sockets, concurrencia, la que permite las múltiples conexiones entre servidor y cliente o entre cliente y cliente.

Durante este desarrollo se enfrentaron desafíos específicos, solo para resumir el principal era el desconocimiento de la programación, el cual mediante la investigación y el apoyo de personas con más experiencia se logró superar, esto es base fundamental para el proceso de crecimiento profesional como futuros desarrolladores.

Bibliografía

Docs Python. (23 de 05 de 2024). Obtenido de https://docs.python.org/es/3/howto/sockets.html

El programador Chapuzas. (03 de 10 de 2022). Obtenido de https://programacionpython80889555.wordpress.com/2022/10/03/programacion-con-sockets-en-python-i-estableciendo-conexion-cliente-sevidor/

https://docs.python.org/es/3/. (03 de 05 de 2024). Obtenido de https://docs.python.org/es/3/

IBM. (07 de 05 de 2024). Obtenido de IBM: https://www.ibm.com/docs/es/i/7.5?topic=programming-how-sockets-work

Pino, D. (05 de 06 de 2023). *Learn Microsoft*. Obtenido de https://learn.microsoft.com/es-es/dotnet/fundamentals/networking/sockets/socket-services