Juan Malo, Roberto Romero

Informe sobre la Implementación del Sistema Cliente-Servidor usando sockets

Introducción

Este informe describe la implementación de un sistema de comunicación centralizado cliente-servidor, en el cual varios clientes se conectan a un servidor para enviar y recibirmensajes de manera eficiente y segura. Se implementa un protocolo de comunicación que asegura la integridad de los mensajes mediante el uso de HMAC (Hash-based Message Authentication Code). Además, se incluye una interfaz gráfica amigable que permite a los usuarios conectarse al servidor, enviar y recibir mensajes, y manejar las desconexiones y reconexiones automáticas de manera eficiente.

Requisitos del sistema

Los principales requisitos funcionales y técnicos del sistema son los siguientes:

- Arquitectura centralizada: Varias instancias de clientes deben poder conectarse aun servidor central.
- Comunicación mediante sockets TCP: El sistema debe utilizar sockets TCP parapermitir la comunicación bidireccional entre el cliente y el servidor.
- Protocolo de comunicación seguro: Debe garantizarse la integridad de los mensajes transmitidos entre el cliente y el servidor.
- Manejo de conexiones y desconexiones: El sistema debe permitir que los clientesse reconecten automáticamente si se desconectan de la red.
- Interfaz de usuario amigable: El sistema debe incluir una interfaz gráfica sencillapara la interacción con los usuarios.

Implementación en Python

El sistema está compuesto por dos componentes principales: un servidor que gestiona las conexiones de los clientes y maneja la retransmisión de mensajes, y un cliente que seconecta al servidor, envía mensajes y los recibe en una interfaz gráfica.

Arquitectura Cliente-Servidor

El sistema sigue una arquitectura centralizada de tipo cliente-servidor, donde el servidoractúa como el nodo central que maneja las conexiones de múltiples clientes. Cada cliente se conecta al servidor utilizando un socket TCP y se comunica a través de este canal.

Juan Malo, Roberto Romero

Servidor

El servidor escucha en una dirección IP y puerto específicos, aceptando conexiones entrantes y gestionando las comunicaciones de los clientes conectados. Cada cliente esmanejado en un hilo separado, lo que permite la atención de múltiples clientes simultáneamente.

```python
def start_server():
    host = '127.0.0.1'
    port = 5555

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((host, port))
    server.listen(5)

    # Cargar los nicknames de los clientes desde el archivo
    global clients

    print(f"Servidor seguro iniciado en {host}:{port}")

    while True:
        try:
            client_socket, addr = server.accept()
            print(f"Conexión segura desde {addr}")
            threading.Thread(target=handle_client, args=(client_socket, addr), daemon=True).start()
        except Exception as e:
            print(f"Error al aceptar una nueva conexión: {e}")
        """except KeyboardInterrupt:
            print("Servidor detenido.")
            break"""
```

Este bloque de código establece que el servidor escuche en la IP local (127.0.0.1) en elpuerto 5555 y acepte hasta 5 conexiones simultáneas.

Cliente

Cada cliente se conecta al servidor y establece una comunicación segura mediante el usode sockets TCP. Además, los clientes pueden enviar y recibir mensajes al servidor de forma eficiente.

```python
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((host, port))

print("Conexión segura establecida con el servidor.")
```

En este fragmento del código, el cliente crea un socket TCP y se conecta al servidorutilizando la IP y puerto establecidos.

Protocolo de Comunicación Seguro

El sistema implementa un protocolo de comunicación seguro que garantiza la integridadde los mensajes mediante el uso de HMAC. Cada mensaje que se envía al servidor es

acompañado de su correspondiente HMAC, generado utilizando una clave secreta compartida entre el servidor y el cliente. El servidor, al recibir el mensaje, verifica la validezdel HMAC para asegurarse de que el mensaje no haya sido alterado.

Generación y Verificación de HMAC

La función generate_hmac genera el código de autenticación del mensaje, y verify_hmacverifica que el mensaje recibido sea íntegro:

```python
def generate_hmac(message):
    return hmac.new(secret_key, message.encode('utf-8'), hashlib.sha256).hexdigest()

# Función para verificar la integridad de un mensaje usando HMAC
def verify_hmac(message, received_hmac):
    calculated_hmac = generate_hmac(message)
    return hmac.compare_digest(calculated_hmac, received_hmac)
```

La verificación de la integridad de los mensajes se realiza al comparar el HMAC calculado con el recibido. Si los valores coinciden, el mensaje es considerado válido. De lo contrario,se descarta el mensaje.

Manejo de Conexiones y Desconexiones

El sistema está diseñado para manejar las conexiones y desconexiones de manera eficiente. Cuando un cliente se desconecta, el servidor elimina la referencia a este clientey actualiza la lista de usuarios conectados. Además, el servidor permite que los clientes sereconecten automáticamente en caso de que la conexión se pierda.

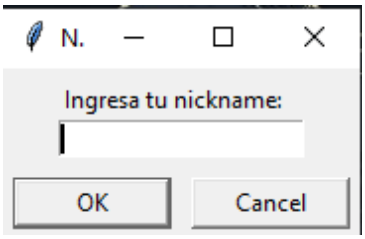
En el cliente, se implementa un mecanismo de reconexión automática en caso de que elcliente pierda la conexión con el servidor. La función reconnect_to_server se encarga de reintentar la conexión cada 5 segundos:

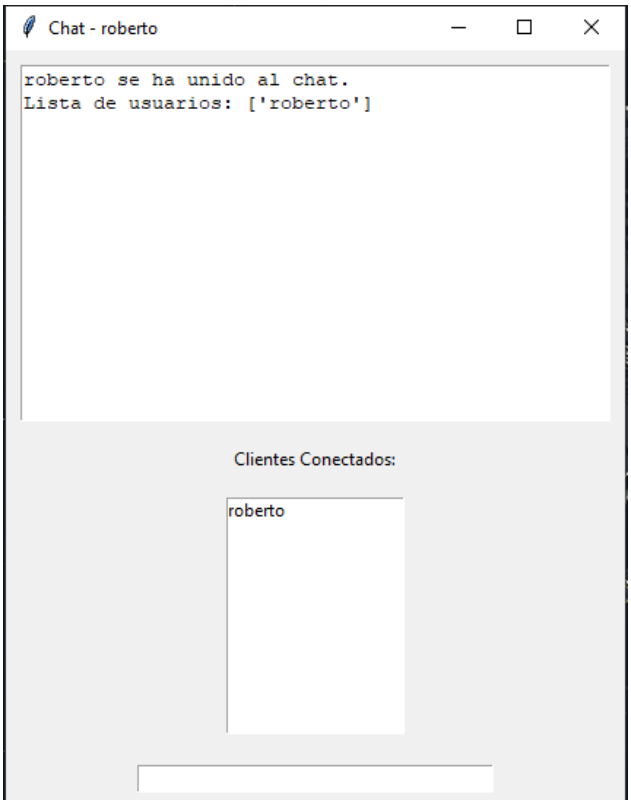
```python
def reconnect_to_server(add_message_callback: callable) -> None:
    global nickname
    global client_socket

    while True:
        try:
            print("Intentando conectar al servidor...")
            host = '127.0.0.1'
            port = 5555

            # Crear contexto SSL
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client_socket.connect((host, port))

            print("Conexión segura establecida con el servidor.")

            client_socket.send(nickname.encode('utf-8'))

            # Iniciar un hilo para recibir mensajes
            threading.Thread(target=receive_messages, args=(client_socket, add_message_callback), daemon=True).start()

            return

        except Exception as e:
            print(f"Error de conexión: {e}")
            print("Reintentando en 5 segundos...")
            time.sleep(5)
```

Este bloque de código asegura que si un cliente se desconecta o experimenta problemasde red, intentará volver a conectarse al servidor automáticamente.

Interfaz de Usuario Amigable

El cliente implementa una interfaz gráfica utilizando la biblioteca tkinter, que proporcionauna experiencia de usuario sencilla y accesible. Los usuarios pueden ingresar su nickname, escribir y enviar mensajes, y ver la lista de clientes conectados. La interfaz también muestra los mensajes recibidos en tiempo real.





Comunicación entre Clientes y Servidor

El servidor retransmite los mensajes de un cliente a todos los demás clientes conectadosmediante la función broadcast. Los mensajes son enviados con su HMAC correspondiente, garantizando que no hayan sido alterados.

```python
def broadcast(message, sender_socket=None):
    for nickname, client_socket in clients.items():
        if client_socket != sender_socket:  # No enviar el mensaje al remitente
            try:
                hmac_hash = generate_hmac(message)
                client_socket.send(f"{message}|{hmac_hash}".encode('utf-8'))
            except Exception as e:
                print(f"Error al enviar mensaje a {nickname}: {e}")
                client_socket.close()
                del clients[nickname]
```

Implementación en Node.js

Arquitectura Centralizada, de Tipo Cliente-Servidor

El sistema propuesto sigue una arquitectura centralizada de tipo cliente-servidor, donde un servidor único centraliza la comunicación y gestiona las conexiones de varios clientes.

Este modelo garantiza una comunicación ordenada y eficiente entre los participantes. El servidor actúa como intermediario para el envío y la recepción de mensajes, mientras quelos clientes se conectan a él para interactuar entre sí.

En la implementación del servidor (código servidor.js), se utiliza net.createServer() paracrear un servidor que escucha en el puerto 8000, permitiendo que múltiples clientes se conecten simultáneamente. Cada cliente que se conecta obtiene un socket único, a travésdel cual se gestiona la comunicación. Este servidor maneja conexiones persistentes de forma eficiente y maneja la desconexión y reconexión de los clientes.

Por otro lado, en el cliente (código cliente.js), se usa net.createConnection() paraestablecer una conexión con el servidor. La IP y el puerto del servidor están preconfigurados como SERVER_HOST = '127.0.0.1' y SERVER_PORT = 8000, asegurandoque cada cliente se conecte al servidor adecuado. La interacción entre el cliente y el servidor es bidireccional, permitiendo el intercambio de mensajes en tiempo real.

a) Implementación de Sockets TCP

La comunicación entre los clientes y el servidor se realiza mediante sockets TCP, un protocolo fiable que garantiza que los mensajes se entreguen de manera correcta entre laspartes.

- Servidor: El servidor, al crear una instancia con net.createServer(), abre un socketTCP para cada cliente que se conecta. Cada vez que un cliente envía un mensaje, este se procesa a través del socket correspondiente. El evento socket.on('data') se utiliza para recibir los datos del cliente, y socket.write() permite enviar respuestas alcliente. Esta implementación asegura que los datos se transmitan de manera ordenada y confiable entre los clientes y el servidor.
- Cliente: En el cliente, la función net.createConnection() establece un socket que seconecta al servidor. La comunicación se realiza mediante client.write() para enviarmensajes cifrados al servidor y client.on('data') para recibir las respuestas. Los datos que se envían están cifrados, garantizando la confidencialidad de la comunicación. Los sockets TCP aseguran que los mensajes se entreguen sinpérdidas de datos o corrupciones.

b) Protocolo de Comunicación Seguro: Verificación de Integridad de los Mensajes yCifrado

La seguridad de la comunicación se garantiza mediante el uso de cifrado simétrico y autenticación de mensajes con HMAC (Hash-based Message Authentication Code). Estoasegura que los mensajes no sean interceptados ni modificados durante su transmisión.

- **Cifrado de Mensajes**: El cifrado se realiza utilizando el algoritmo aes-256-cbc, unestándar de cifrado simétrico robusto. En el código del cliente, la función encryptMessage(message) cifra el mensaje antes de enviarlo al servidor. El mensajees transformado en un formato hexadecimal y transmitido de manera segura. El usode una clave secreta de 32 bits (KEY = '11111111111111111111111111111111') y un vector de inicialización de 16 bits (IV = Buffer.from('0000000000000000')) asegura que el cifrado sea seguro y único para cada sesión de comunicación.
- **Descifrado de Mensajes**: En el servidor, el mensaje cifrado es descifrado mediante la función decryptMessage(encryptedMessage). Esta función utiliza el mismo algoritmo y parámetros de clave e IV para garantizar que el mensaje originalsea recuperado correctamente. Al usar el mismo mecanismo de cifrado, tanto el cliente como el servidor pueden intercambiar mensajes de manera segura.
- **Verificación de Integridad con HMAC**: La integridad de los mensajes se asegura mediante el uso de HMAC. En el cliente, la función generateHMAC(message) generaun código HMAC para cada mensaje que se envía. Este código es transmitido junto con el mensaje cifrado. En el servidor, la función verifyHMAC(message, hmac) verifica que el HMAC del mensaje recibido coincida con el calculado en el servidor, asegurando que el mensaje no haya sido alterado. Si la verificación falla, se envía un mensaje de error al cliente, garantizando que solo los mensajes legítimos sean procesados.

c) Manejo de Conexiones y Desconexiones de Clientes

El manejo adecuado de las conexiones y desconexiones es esencial para mantener la estabilidad del sistema. El cliente debe poder reconectarse automáticamente si se pierdela conexión, mientras que el servidor debe ser capaz de gestionar la desconexión de los clientes sin causar problemas en la comunicación.

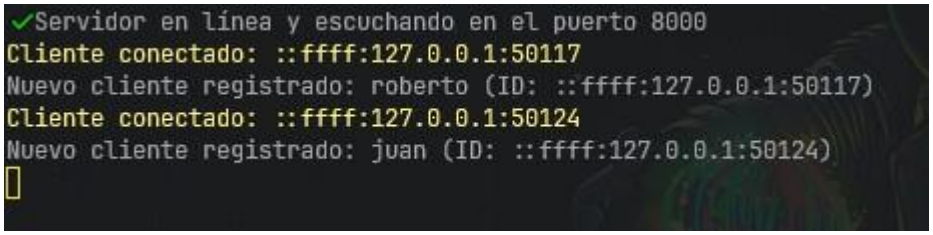
- **Reconexión Automática del Cliente**: En el código del cliente, el manejo de desconexiones se implementa con el evento client.on('close'). Cuando el clientedetecta que la conexión con el servidor se ha perdido, utiliza un temporizador (setTimeout(connectToServer, RECONNECT_INTERVAL)) para intentar reconectarse automáticamente después de un intervalo de 3 segundos. Este enfoque asegura que la pérdida de conexión no interrumpa la comunicación permanentemente.
- **Desconexión y Manejo de Clientes en el Servidor**: En el servidor, cuando un cliente se desconecta, se maneja a través del evento socket.on('close'). El servidorelimina el cliente de la lista de clientes activos (clients.delete(id)) y notifica a todoslos demás clientes sobre la desconexión utilizando la función broadcast. Esto

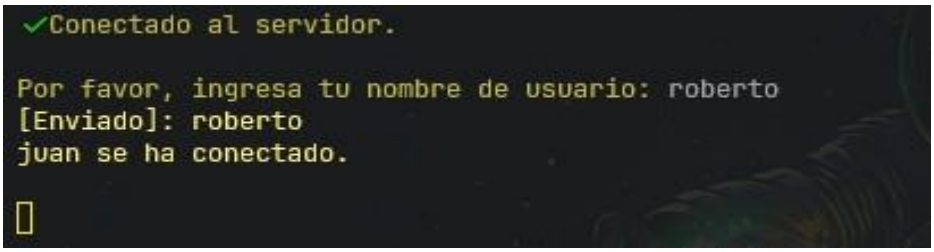
asegura que los demás usuarios sean informados de manera inmediata cuando uncliente se desconecta del sistema.

d) Interfaz de Usuario Amigable

El sistema incluye una interfaz de usuario basada en consola que es sencilla de usar ypermite al usuario interactuar con el servidor de manera eficiente.

- **Ingreso del Nombre de Usuario**: Cuando el cliente se conecta por primera vez, sesolicita al usuario que ingrese su nombre. Este mecanismo permite identificar al cliente y evitar que múltiples clientes usen el mismo nombre. Además, el sistema verifica si el nombre ya está en uso antes de permitir la conexión.
- **Envío y Recepción de Mensajes**: Una vez conectado, el cliente puede enviar yrecibir mensajes de forma continua. El cliente solicita mensajes al usuario mediante la función requestMessage(), que captura la entrada de texto y la envía alservidor. Además, los mensajes recibidos del servidor se muestran en la consola con formato destacado utilizando el paquete chalk para mejorar la legibilidad.
- **Notificaciones de Conexión y Desconexión**: Cuando un cliente se conecta o desconecta, el servidor envía notificaciones a todos los demás clientes. El cliente muestra estos mensajes en la consola, utilizando colores y formato para hacer quelas notificaciones sean fácilmente visibles y comprensibles para el usuario. Esto proporciona una experiencia de usuario clara y amigable.





Link del repositorio: https://github.com/Juanja1306/Sockets