**FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS**

**ESCUELA PROFESIONAL DE INFORMÁTICA**

**SISTEMAS OPERATIVOS II**

“Comunicación entre Procesos – Laboratorio 04”



**AUTOR**

Morales Esquivel Christian Anthony

**DOCENTE**

Mg. Yosip Urquizo Gómez

**CICLO:** VIII

**Trujillo - Perú**

**2024**

**Laboratorio 04**

Ing. Yosip Urquizo Gómez

Desarrollar una aplicación práctica basada en el tema de “**Comunicación entre Procesos”,** podemos proponer la creación de una aplicación sencilla utilizando **Python** con su biblioteca de **sockets**.

Este lenguaje y herramienta permiten implementar un modelo de comunicación entre procesos de manera accesible y fácil de entender, al tiempo que introduce los conceptos clave de comunicación en sistemas distribuidos.

Aquí te dejo una propuesta de ejercicio práctico con Python, donde los estudiantes desarrollarán una aplicación de **chat en red distribuida** usando **sockets TCP/IP**.

**Ejercicio Práctico: Implementación de un Chat entre Procesos con Python (Sockets)**

**Objetivo:**

Los estudiantes crearán una aplicación de **chat entre dos o más procesos** en diferentes maquinas (o en la misma) usando **sockets TCP** en Python. Esta aplicación les permite aprender como enviar y recibir mensajes entre procesos distribuidos.

**Pasos del Ejercicio:**

1. **Conceptos clave a enseñar:**
   * **Socket:** Es un punto final de una conexión de red entre dos procesos.
   * **Servicio y Cliente:** El servidor espera conexiones de los clientes, mientras que los clientes inician la comunicación.
   * **Protocolo TCP/IP:** El protocolo TCP garantiza la entrega de los mensajes en el orden correcto.
   * **Bloqueo/No Bloqueo:** Los estudiantes verán la diferencia entre comunicación síncrona(bloqueo) y asíncrona (no bloqueo) durante el intercambio de mensajes.
2. **Código Propuesto para el Chat entre Procesos:**
3. **Implementación del Servidor:**
4. **Implementación del Cliente**

****

1. **Explicación del Código:**
2. **Servidor:**
   * El servidor se ejecuta en una maquina y escucha conexiones de múltiples clientes.
   * Una vez que se conecta un cliente, se genera un nuevo hijo para gestionar la comunicación con ese cliente, permitiendo que el servidor acepte más conexiones simultáneamente.
   * El servidor recibe mensajes del cliente y los imprime en la consola, luego envía una respuesta al cliente confirmando la recepción del mensaje.
3. **Cliente:**

* El cliente se conecta al servidor utilizando el socket TCP.
* El cliente puede enviar mensajes al servidor, que luego los recibe, procesa y responde.
* La comunicación es continua hasta que el usuario finaliza la conexión (pueden implementarse condiciones de salida como un mensaje “exit”).

1. **Extensiones del Proyecto (Mejoras Opcionales):**
2. **Multicliente (Broadcast):** El servidor puede difundir los mensajes recibidos de un cliente a todos los demás clientes conectados, simulando un **chat grupal**.
3. **Mensajes encriptados:** Usar una capa de cifrado (como SSL/TLS) para que los mensajes entre cliente y servidor sean seguros.
4. **Comunicación asíncrona:** Implementar una versión en la que los clientes no queden bloqueados esperando la respuesta del servidor, utilizando bibliotecas como **asyncio**.
5. **Interfaces gráficas(opcional):** Utilizar bibliotecas como **TKinter** o **PyQt** para crear una interfaz gráfica para el chat.
6. **Retos Adicionales para los Estudiantes:**
7. **Manejo de fallos:**
   * ¿Qué ocurre si el cliente se desconecta abruptamente? Pueden modificar el código para gestionar estas situaciones.
     + Si un cliente se desconecta sin previo aviso, el servidor detectará un error en la recepción de mensajes y lo manejará cerrando el socket correspondiente. Los demás clientes no se verán afectados, pero no se recibe ningún aviso formal del cliente desconectado.
8. **Sincronización:**
   * ¿Cómo podríamos sincronizar varios clientes para que todos reciban mensajes en el mismo orden? Explorar la sincronización de procesos y mensajes.
     + Podríamos implementar un identificador de secuencia en cada mensaje que el servidor asigna y utiliza para transmitir los mensajes en orden. Alternativamente, se podría usar un algoritmo de consenso distribuido (como Paxos o Raft) para garantizar el orden de entrega.
9. **Detección de latencia:**
   * ¿Cómo afecta la latencia de red en la comunicación entre los procesos distribuidos?
     + La latencia de red puede hacer que los mensajes lleguen en un orden distinto al enviado, lo cual podría causar confusión en el contexto de un chat. Para mitigar esto, se pueden usar marcas de tiempo y algoritmos de ordenación para reordenar los mensajes antes de mostrarlos.
10. **Interfaces gráficas(opcional):** 
    * Implementar un sistema donde ciertos mensajes tengan mayor prioridad que otros. (Ejemplo: mensajes de emergencia en un sistema de comunicación de autos autónomos).
      + Se podría agregar un campo de prioridad a cada mensaje y modificar el servidor para que difunda los mensajes con mayor prioridad antes que otros. Por ejemplo, usando una cola de prioridad.
11. **Conclusión y Aprendizaje Clave:**

Al finalizar el ejercicio, los estudiantes habrán aprendido:

* Cómo enviar y recibir mensajes entre procesos distribuidos utilizando sockets.
* La diferencia entre comunicación síncrona y asíncrona.
* Cómo implementar la concurrencia en el servidor utilizando hilos.
* Como gestionar múltiples conexiones de clientes de forma simultánea.

Esta actividad, además de introducir a los estudiantes en la comunicación entre procesos, también les brinda una base para entender como funcionan aplicaciones más complejas, como sistemas de mensajería instantánea, servidores de juegos en red o aplicaciones loT.

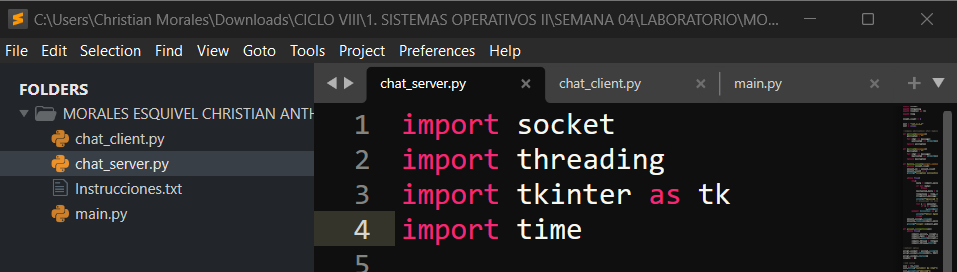
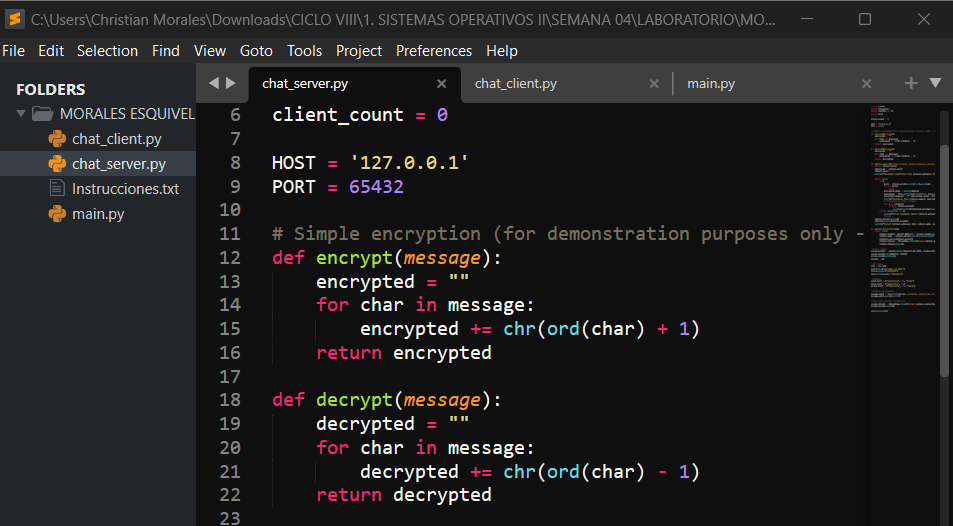
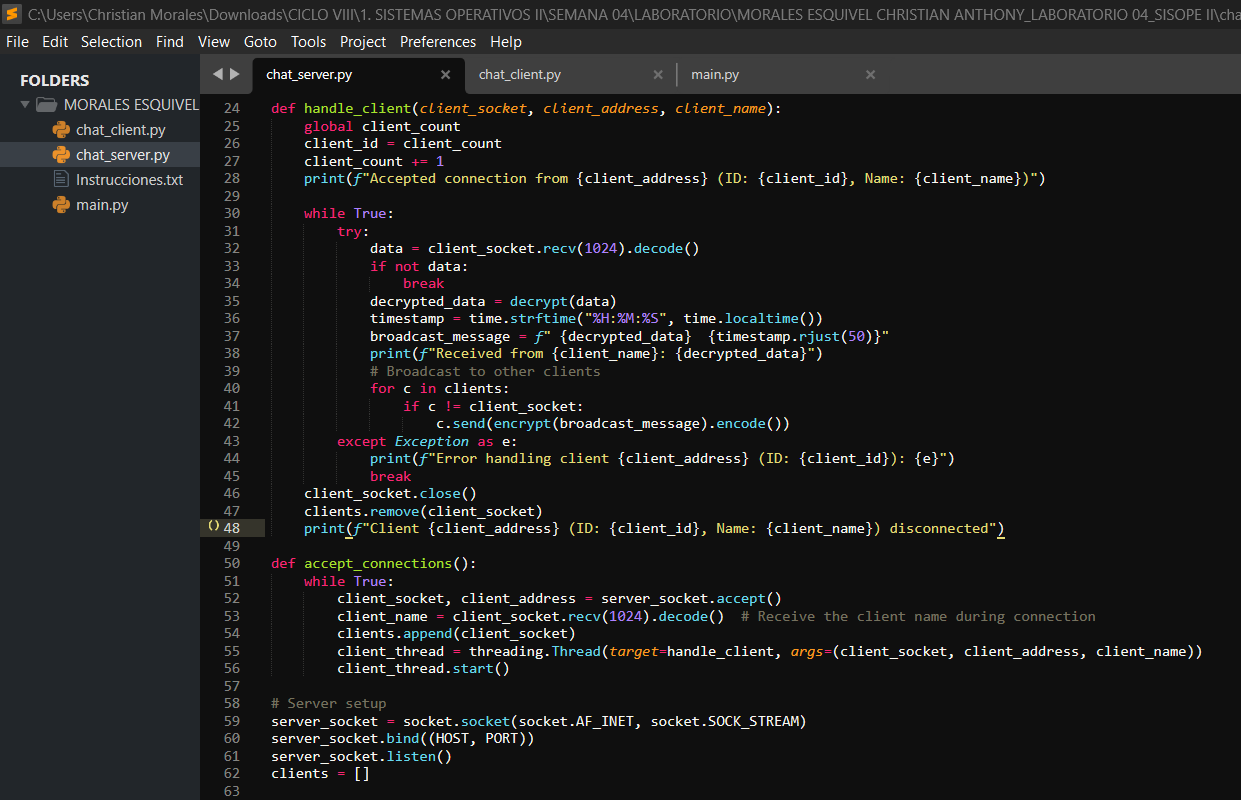
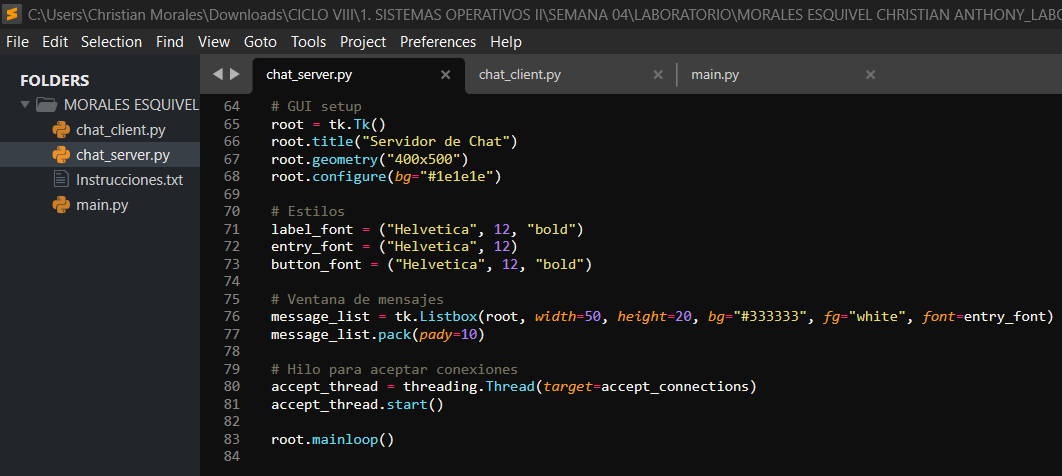
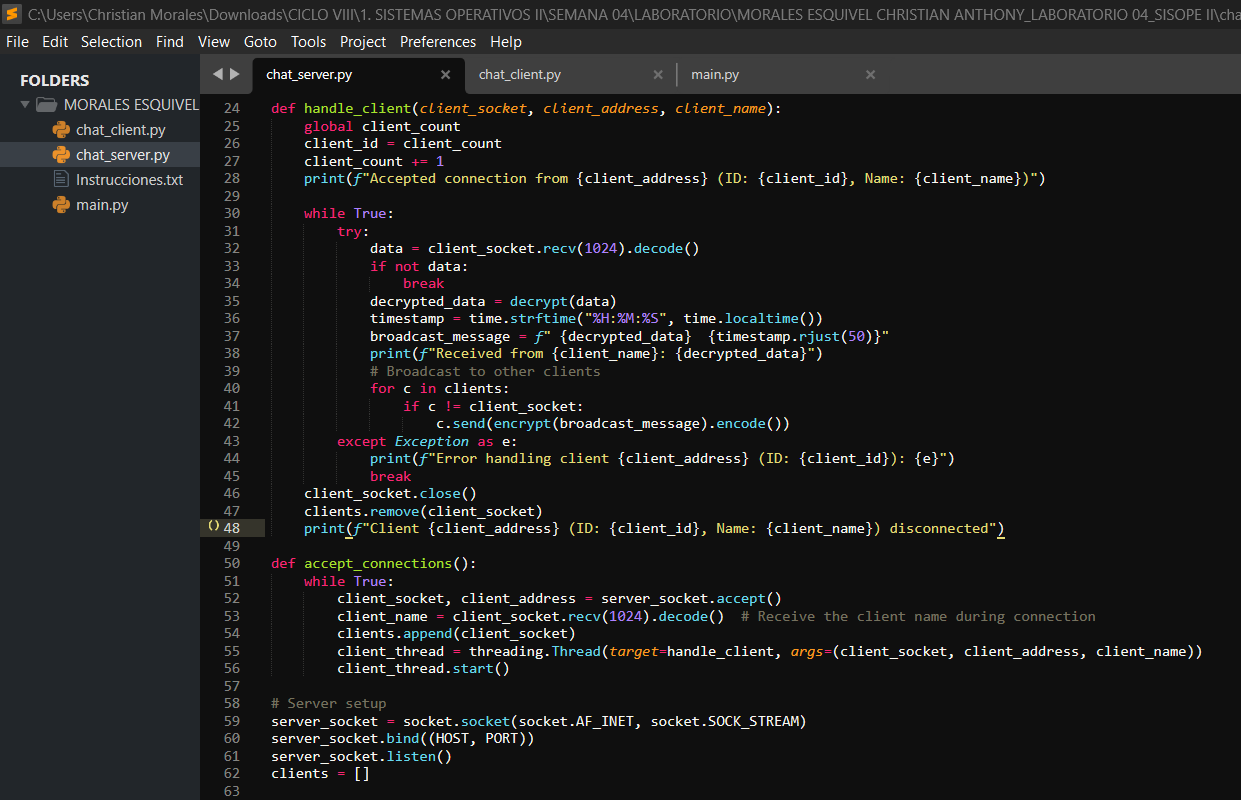
**Explicación del Desarrollo de Ejercicio Práctico:**

**“Implementación de un Chat entre Procesos con Python (Sockets)”**

**Explicación del Código:**

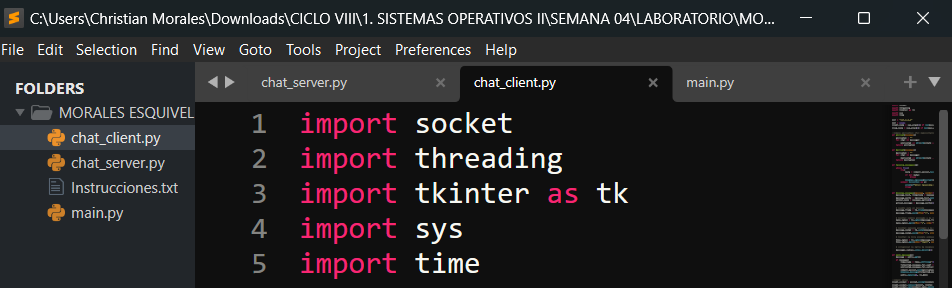
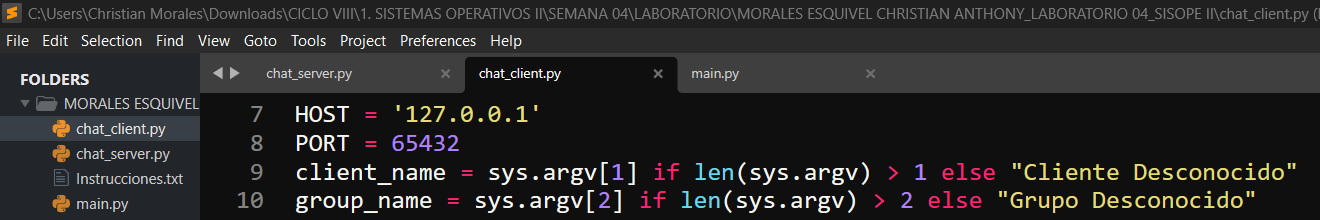
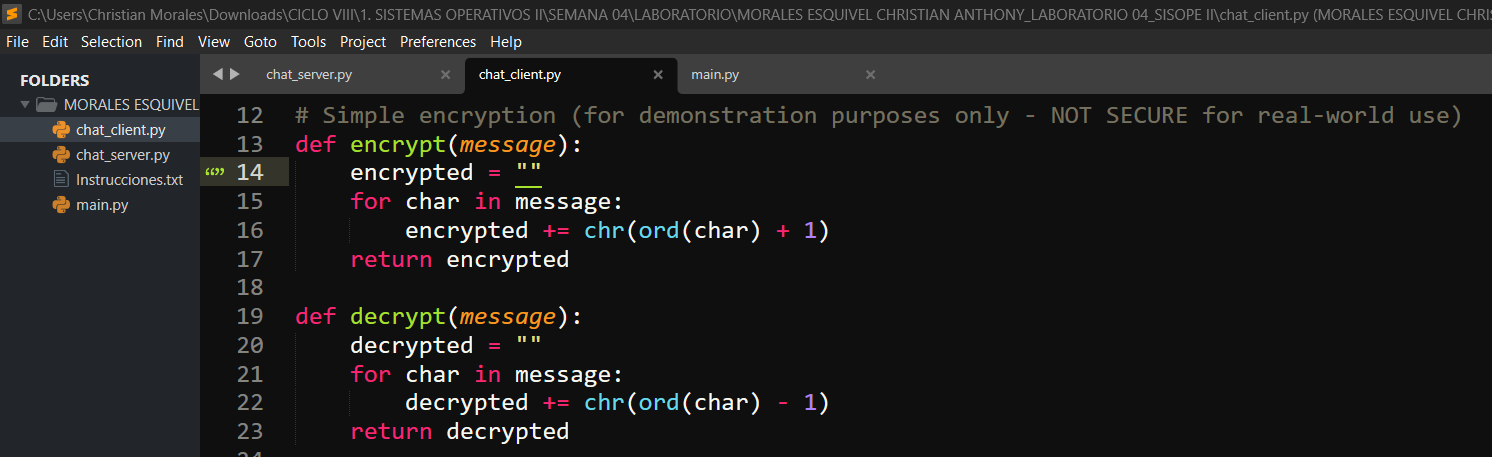
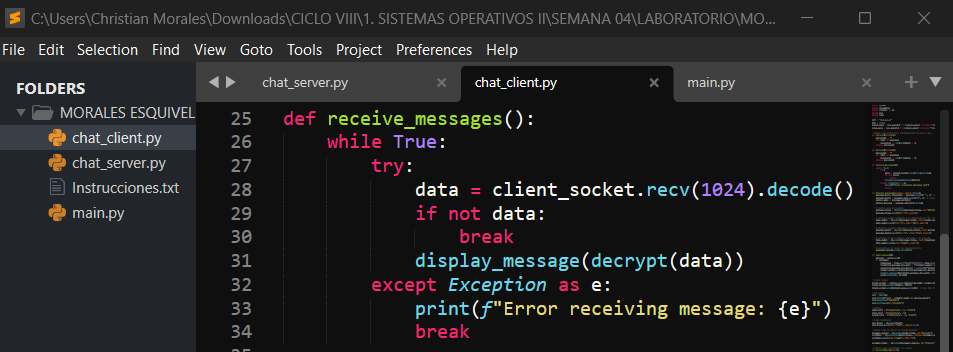
**Archivo 1: chat\_server.py (Servidor)**

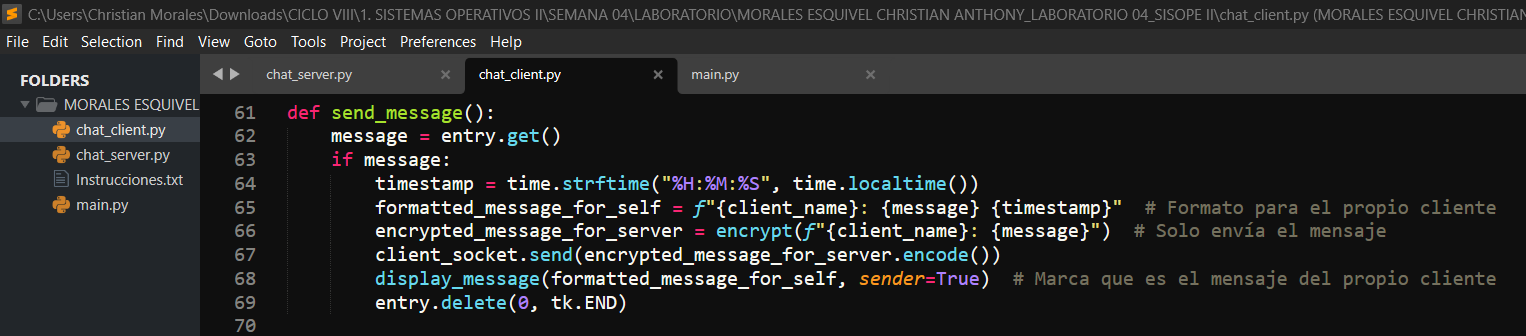
El archivo del servidor implementa un chat grupal que permite a varios clientes conectarse y comunicarse simultáneamente. A continuación, te explico cada parte del código:

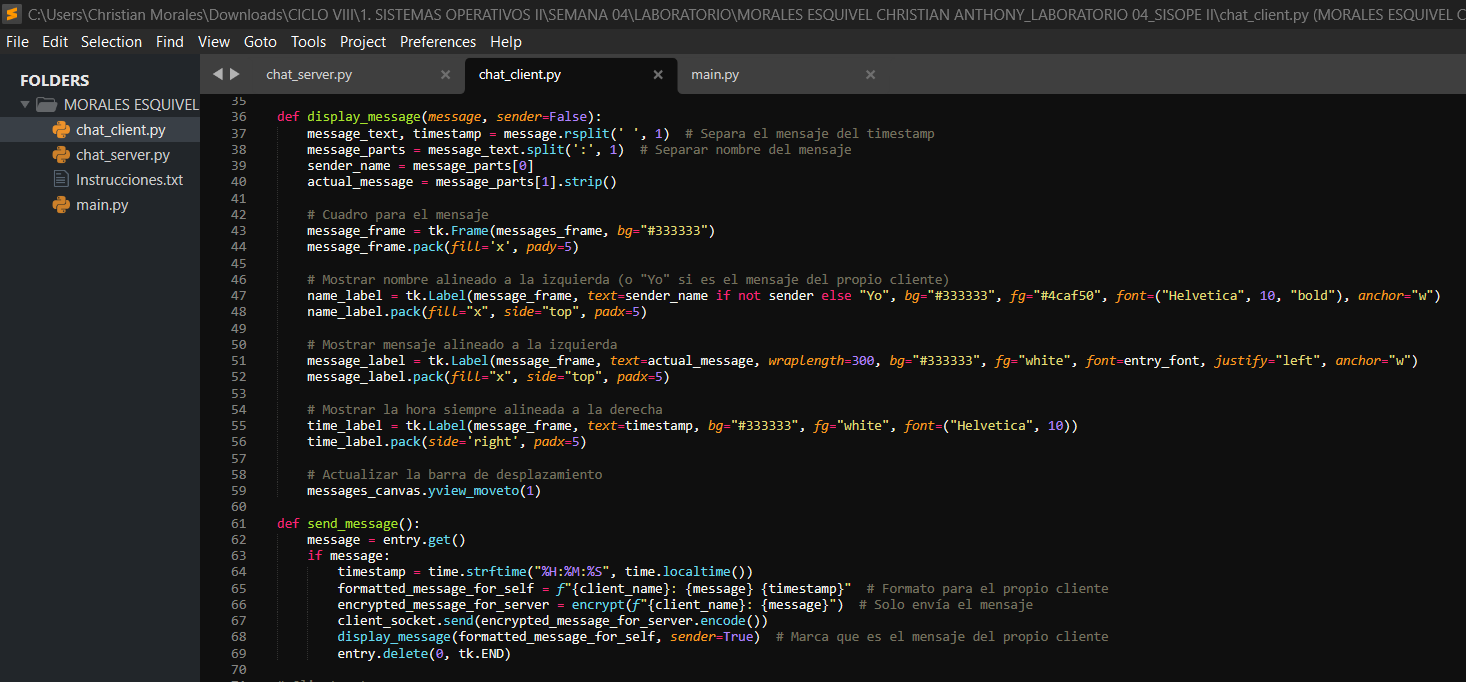
1. **Importación de Bibliotecas**:
   * socket: Para establecer conexiones TCP/IP entre el servidor y los clientes.
   * threading: Para manejar múltiples conexiones simultáneamente usando hilos.
   * tkinter: Para crear la interfaz gráfica del servidor.
   * time: Para registrar la hora en los mensajes.
2. **Configuración Inicial**:
   * El servidor se ejecuta en la dirección 127.0.0.1 (localhost) y en el puerto 65432.
   * Se utiliza un esquema simple de cifrado para encriptar y desencriptar mensajes usando las funciones encrypt y decrypt. Este esquema simplemente desplaza el valor ASCII de cada carácter.
3. **Manejo de Clientes**:
   * El servidor acepta conexiones de múltiples clientes y asigna un identificador único (client\_id) para cada cliente conectado.
   * Cada conexión se gestiona en un hilo independiente usando threading.Thread, lo que permite manejar múltiples clientes sin que el servidor se bloquee.
4. **Función handle\_client**:
   * Recibe los mensajes del cliente y los desencripta usando la función decrypt.
   * Se añade una marca de tiempo (timestamp) a cada mensaje.
   * Luego, envía el mensaje recibido a todos los demás clientes conectados (excepto el que lo envió), utilizando la función de difusión (broadcast).
5. **Interfaz Gráfica**:
   * El servidor cuenta con una sencilla interfaz gráfica creada con Tkinter, que muestra los mensajes en una ventana.
   * La ventana principal contiene un Listbox donde se muestra el historial de mensajes.
6. **Aceptación de Conexiones**:
   * La función accept\_connections está dedicada a aceptar nuevas conexiones y asignar hilos independientes a cada cliente.

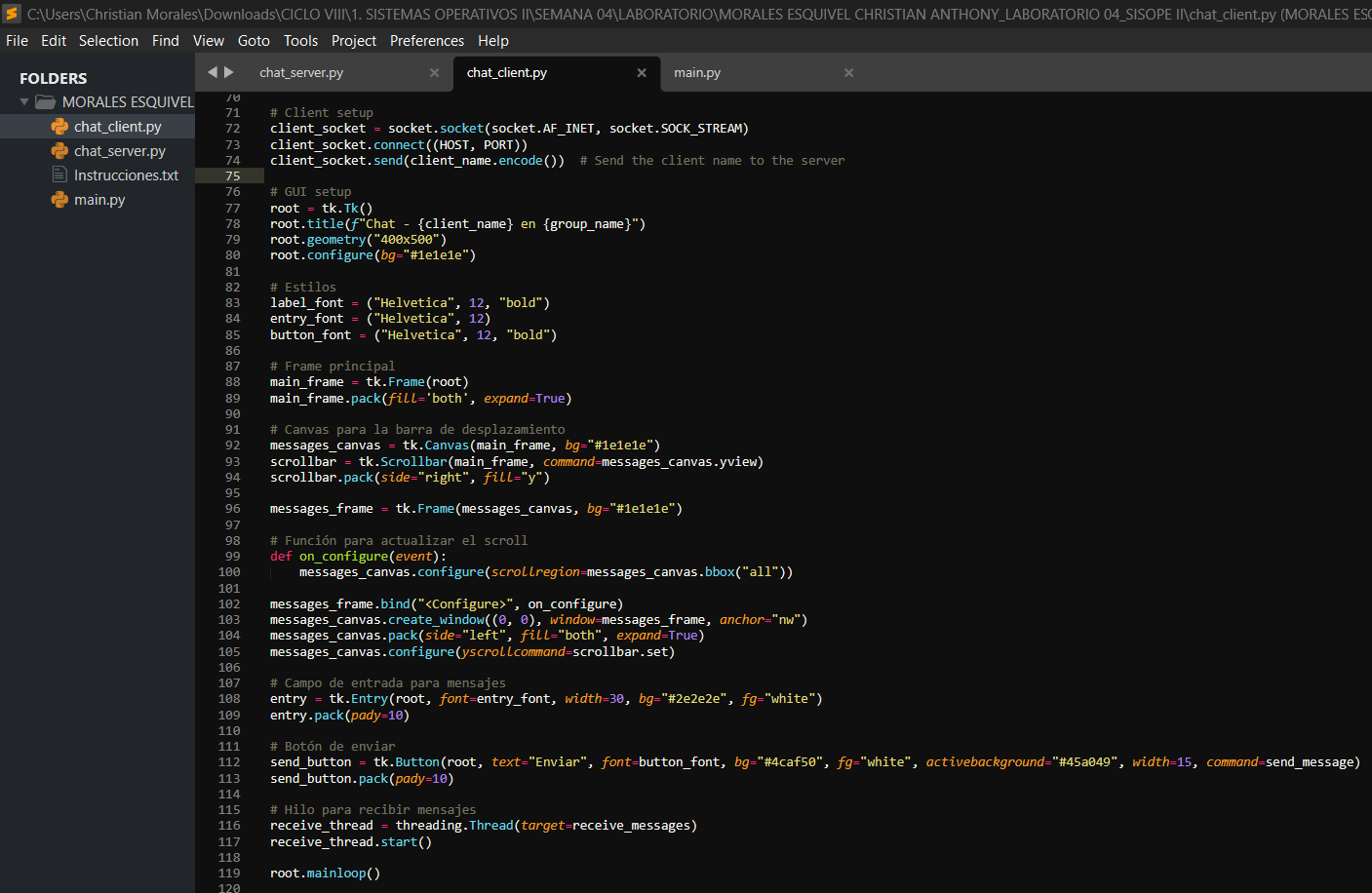
**Archivo 2: chat\_client.py (Cliente)**

El archivo del cliente establece la comunicación con el servidor y permite a cada cliente enviar y recibir mensajes en tiempo real.

1. **Importación de Bibliotecas**:
   * socket: Para conectarse al servidor usando TCP/IP.
   * threading: Para recibir mensajes en un hilo separado, evitando que la interfaz se bloquee.
   * tkinter: Para crear la interfaz gráfica del cliente.
   * sys: Para capturar los argumentos del nombre del cliente y del grupo.
   * time: Para manejar las marcas de tiempo.
2. **Configuración Inicial**:
   * El cliente se conecta a 127.0.0.1 (localhost) en el puerto 65432.
   * El nombre del cliente y el grupo se obtienen de los argumentos de la línea de comandos (sys.argv).
3. **Funciones de Encriptación**:
   * Similar al servidor, se utilizan las funciones encrypt y decrypt para cifrar y descifrar los mensajes enviados y recibidos.
4. **Recepción de Mensajes**:
   * La función receive\_messages recibe mensajes en un hilo separado para que la interfaz gráfica permanezca reactiva.
   * Los mensajes recibidos se desencriptan y se muestran en la ventana del chat.
5. **Envío de Mensajes**:
   * La función send\_message toma el mensaje del campo de entrada, lo encripta y lo envía al servidor.
   * Luego, muestra el mensaje en la propia ventana del cliente (indicando que es su propio mensaje).

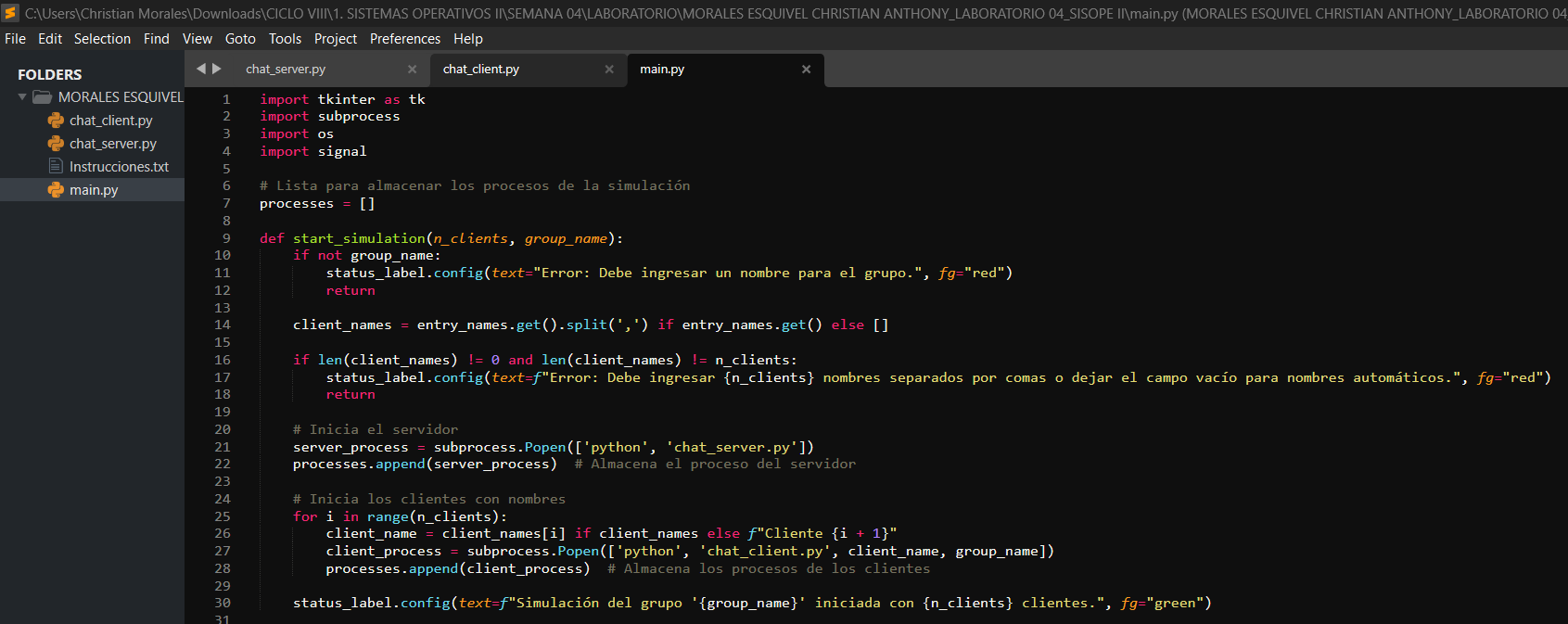


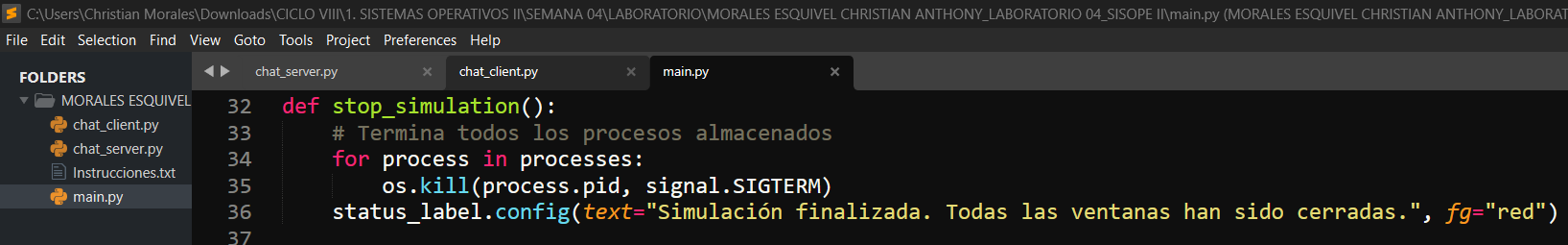
1. **Interfaz Gráfica**:
   * Se utiliza Tkinter para mostrar una interfaz de chat con cuadros de desplazamiento, campos de entrada, y botones para enviar mensajes.
   * El cliente tiene su propio nombre y el nombre del grupo en la ventana del chat.



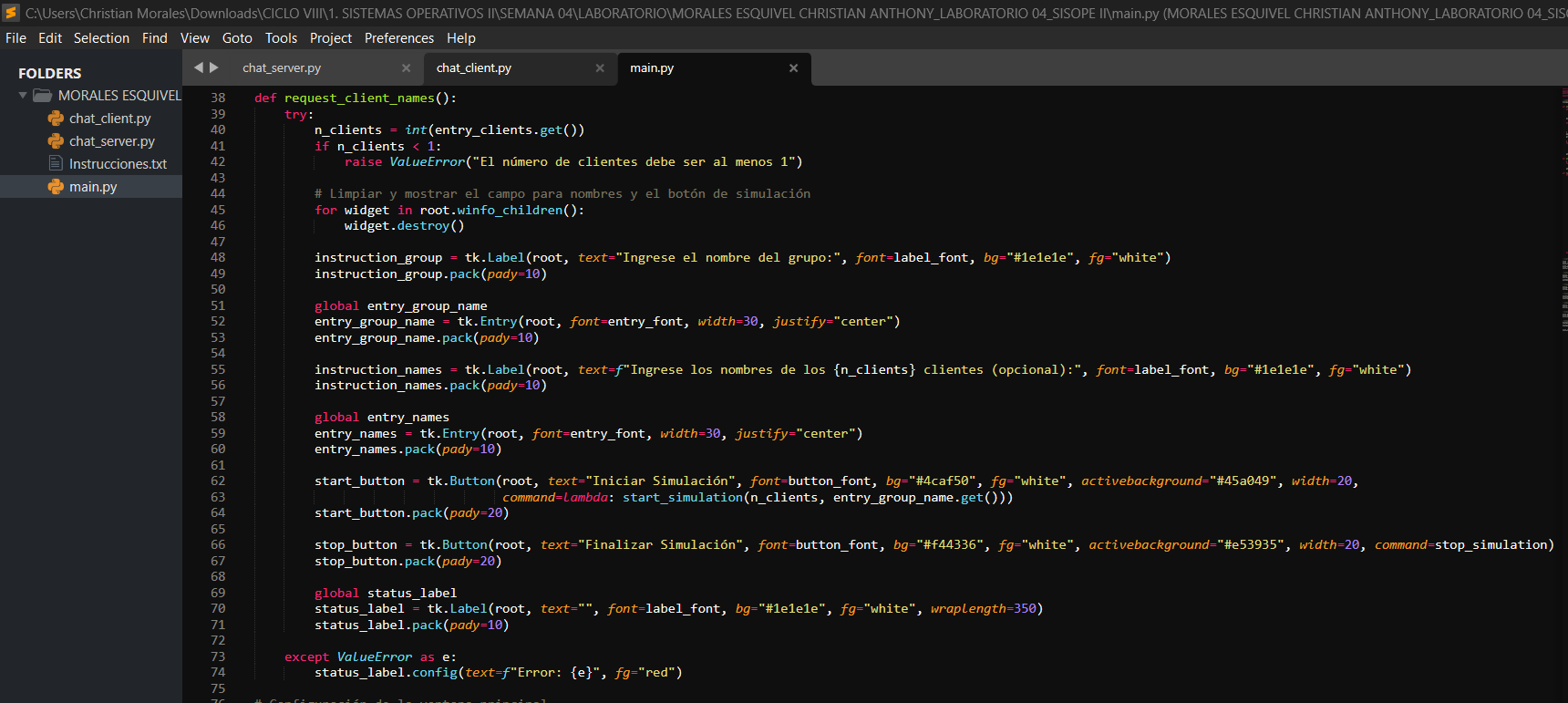
**Archivo 3: main.py (Simulador Principal)**

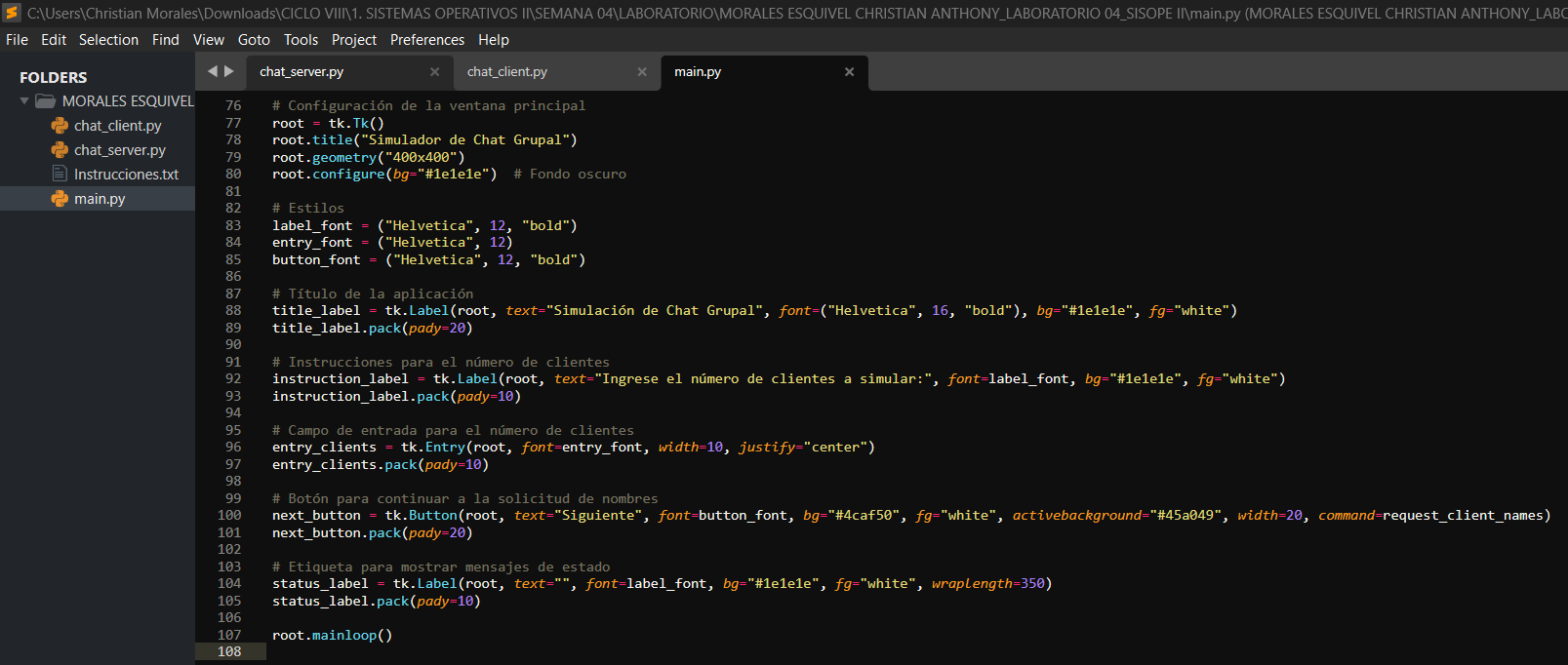
Este archivo es el encargado de gestionar la simulación del chat grupal, permitiendo configurar el número de clientes y administrar el inicio y la finalización del sistema.

1. **Inicio de la Simulación**:
   * La función start\_simulation crea un proceso para el servidor y un proceso independiente para cada cliente utilizando subprocess.Popen.
   * Se asegura de que los clientes tengan nombres únicos y de que el grupo tenga un nombre definido.
2. **Finalización de la Simulación**:
   * La función stop\_simulation termina todos los procesos creados (tanto el servidor como los clientes), cerrando las ventanas asociadas a cada uno.



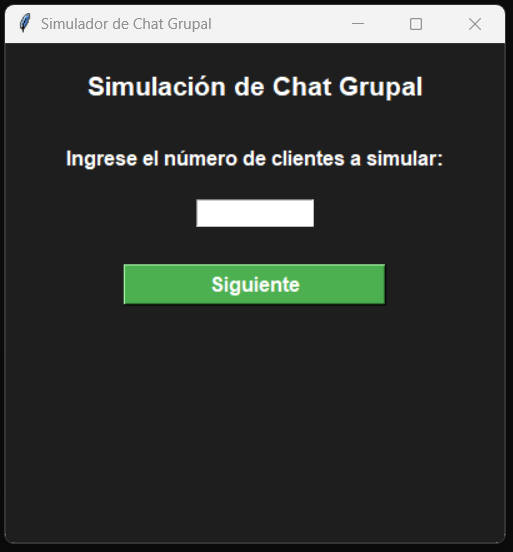
1. **Interfaz de Simulación**:
   * Usa Tkinter para solicitar el número de clientes y permite asignar nombres personalizados a cada uno de ellos.
   * Permite iniciar y finalizar la simulación con botones dedicados (start\_button y stop\_button).
2. **Validación**:
   * Valida la entrada del número de clientes y asegura que el nombre del grupo esté presente.

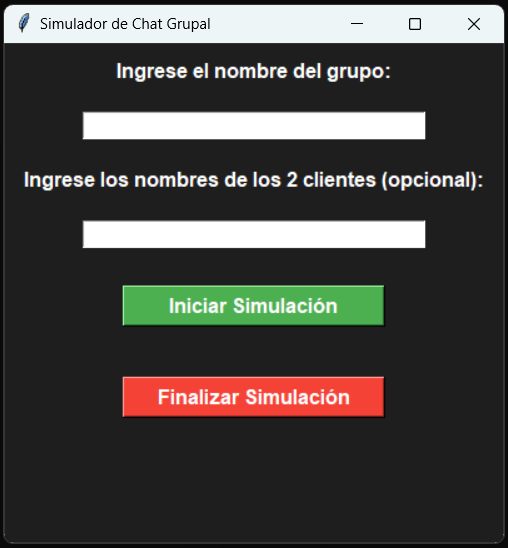




**Ejecución del Código:**

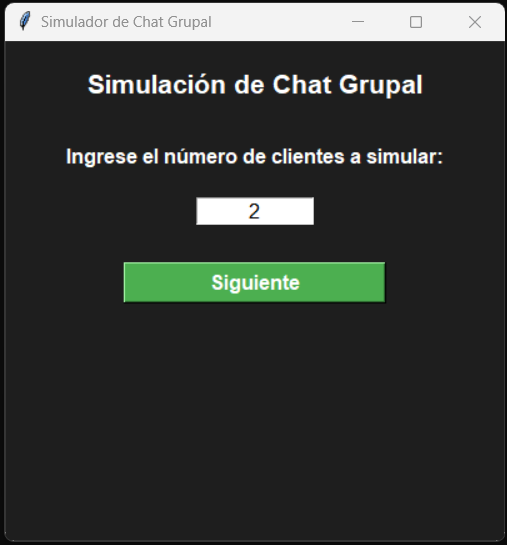
**Ventana Principal**

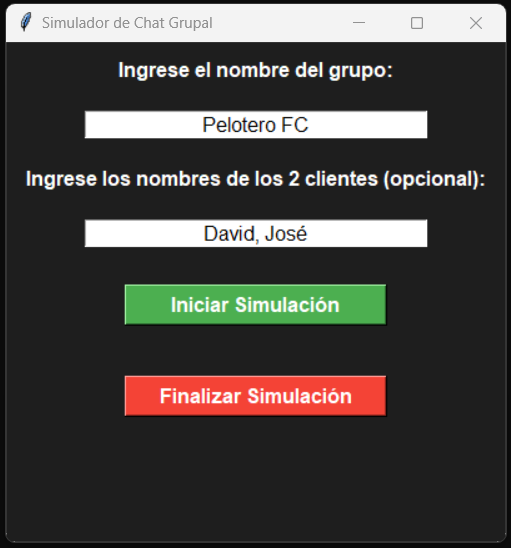
***Vista Principal Parte I***

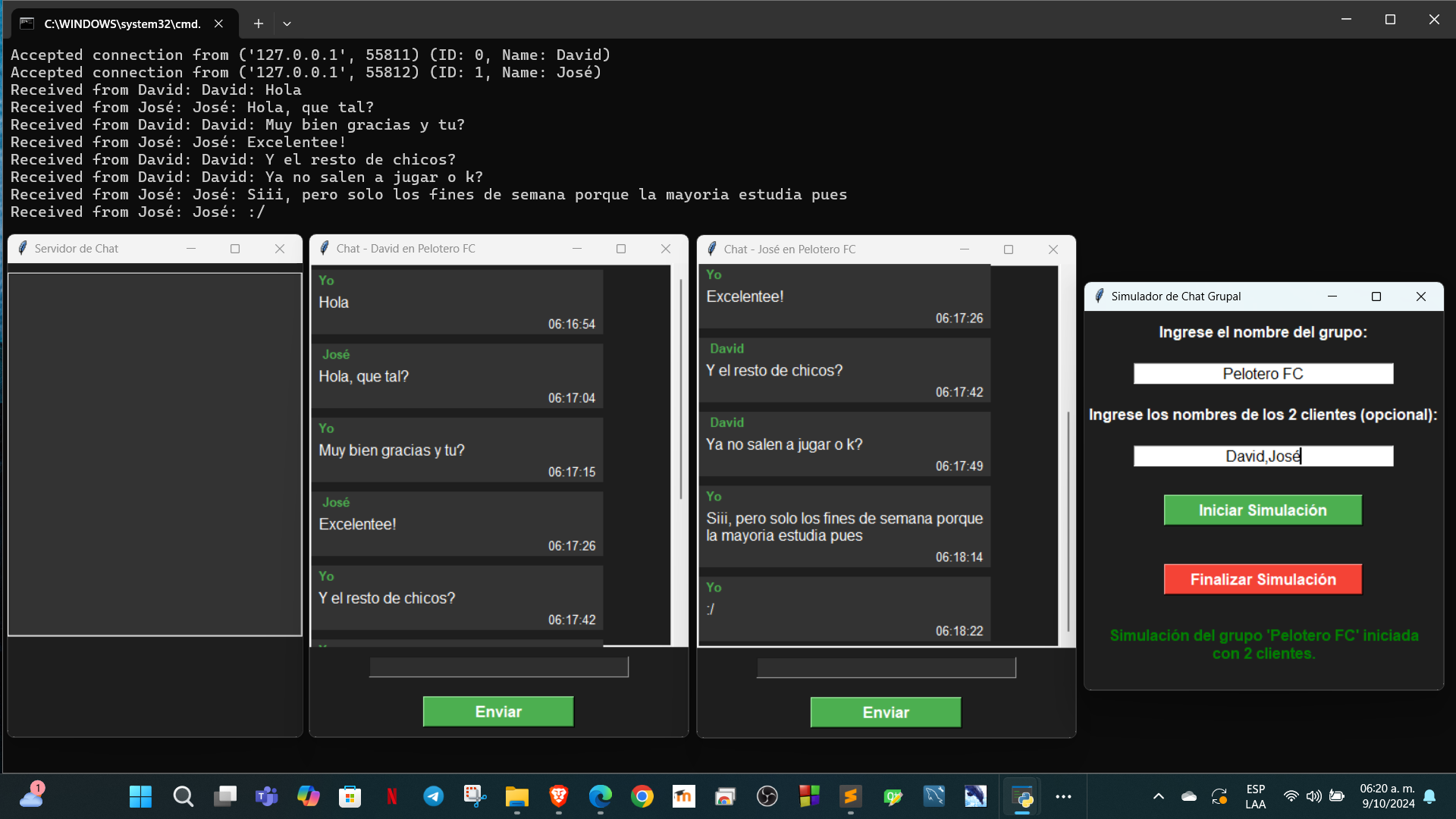
***Vista Principal Parte II***

**Ejecución del Código:**

**Ventana Principal**

***Vista Principal Parte I***

***Vista Principal Parte II***

**Simulación de Chat Grupal**