

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:

A. Implementación del Servidor:

```
import socket
import threading

# Función para manejar múltiples clientes
def handle_client(client_socket):
    while True:
        try:
            # Recibir mensaje del cliente
            message = client_socket.recv(1024).decode('utf-8')
            if not message:
                break
            print(f"Mensaje recibido: {message}")
            # Enviar mensaje de vuelta al cliente
            client_socket.send("Mensaje recibido".encode('utf-8'))
        except:
            print("Error en la conexión")
            break
    client_socket.close()

def start_server():
    # Crear el socket TCP (IPv4, TCP)
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(("0.0.0.0", 5555))
    server_socket.listen(5) # Escuchar hasta 5 conexiones
    print("Servidor iniciado. Esperando conexiones...")

    while True:
        client_socket, addr = server_socket.accept()
        print(f"Conexión establecida con {addr}")
        # Crear un hilo para manejar múltiples clientes
        client_handler = threading.Thread(target=handle_client, args=(client_socket,))
        client_handler.start()

if __name__ == "__main__":
    start_server()
```

B. Implementación del Cliente

```
import socket

def start_client():
    # Crear el socket TCP (IPv4, TCP)
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Conectarse al servidor en la dirección IP y puerto adecuados
    client_socket.connect(("127.0.0.1", 5555))

    print("Conectado al servidor. Puedes enviar mensajes.")
    while True:
        # Leer el mensaje del usuario
        message = input("Tú: ")
        # Enviar el mensaje al servidor
        client_socket.send(message.encode('utf-8'))
        # Recibir la respuesta del servidor
        response = client_socket.recv(1024).decode('utf-8')
        print(f"Servidor: {response}")

if __name__ == "__main__":
    start_client()
```

3. Explicación del Código:

A. 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.

B. 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”).

4. Extensiones del Proyecto (Mejoras Opcionales):

1. **Multicliente (Broadcast):** El servidor puede difundir los mensajes recibidos de un cliente a todos los demás clientes conectados, simulando un **chat grupal**.
2. **Mensajes encriptados:** Usar una capa de cifrado (como SSL/TLS) para que los mensajes entre cliente y servidor sean seguros.
3. **Comunicación asíncrona:** Implementar una versión en la que los clientes no queden bloqueados esperando la respuesta del servidor, utilizando bibliotecas como **asyncio**.
4. **Interfaces gráficas(opcional):** Utilizar bibliotecas como **TKinter** o **PyQt** para crear una interfaz gráfica para el chat.

5. Retos Adicionales para los Estudiantes:

1. 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.

2. 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.

3. 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.

4. 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.

6. 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 IoT.

Explicación del Desarrollo de Ejercicio Práctico:

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

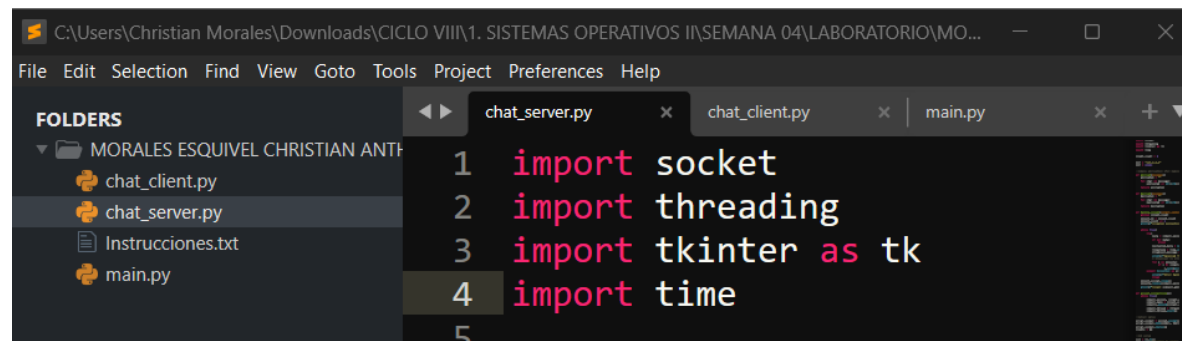
Explicación del Código:

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.

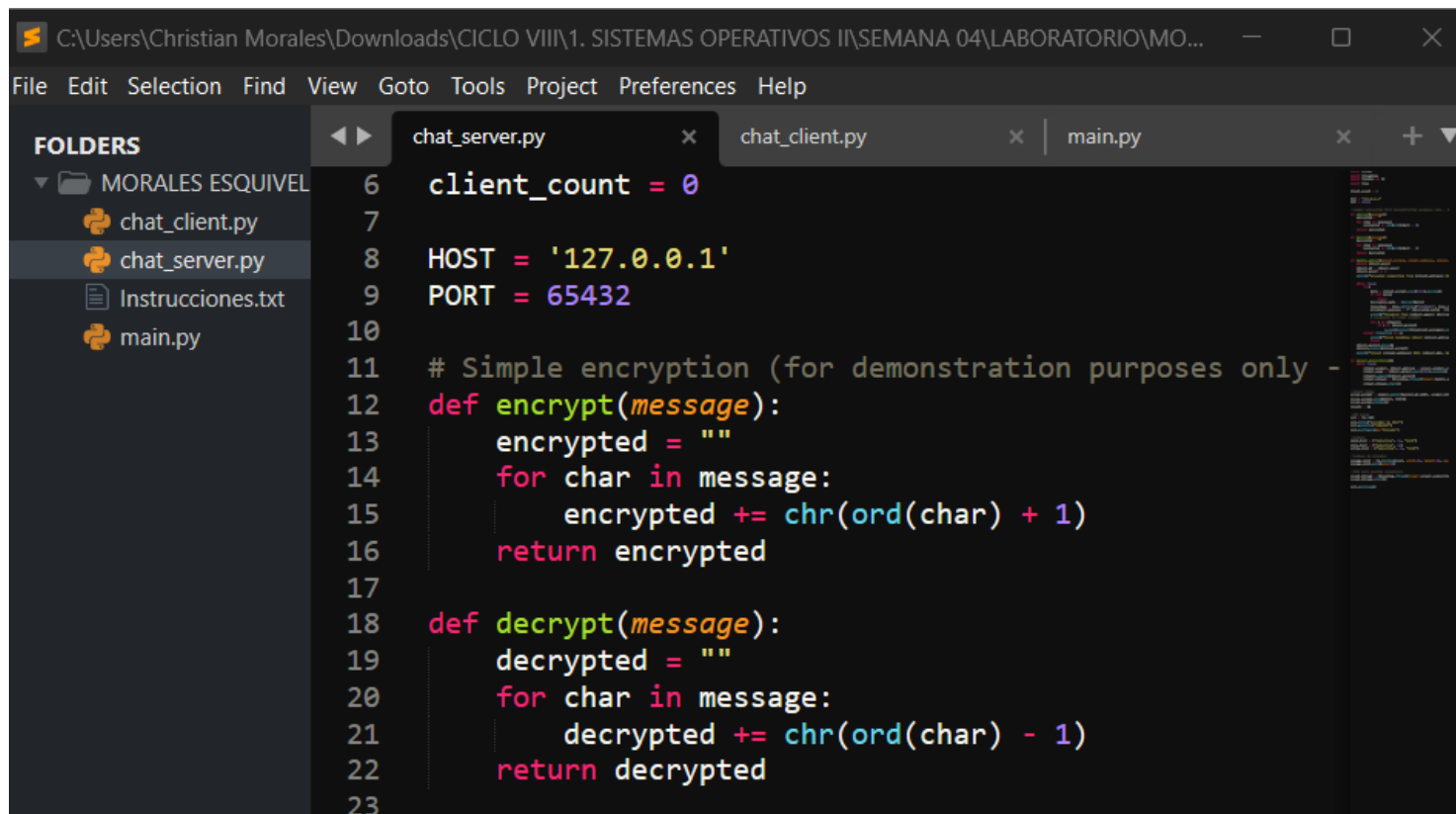


```
1 import socket
2 import threading
3 import tkinter as tk
4 import time
5
```

The screenshot shows a code editor window with a dark theme. The title bar indicates the file path: C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MO... The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The left sidebar shows a folder structure with files chat_client.py, chat_server.py, Instrucciones.txt, and main.py. The main editor area displays the first five lines of code in chat_server.py, which are import statements for socket, threading, tkinter (aliased as tk), and time.

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.



```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MO...
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py

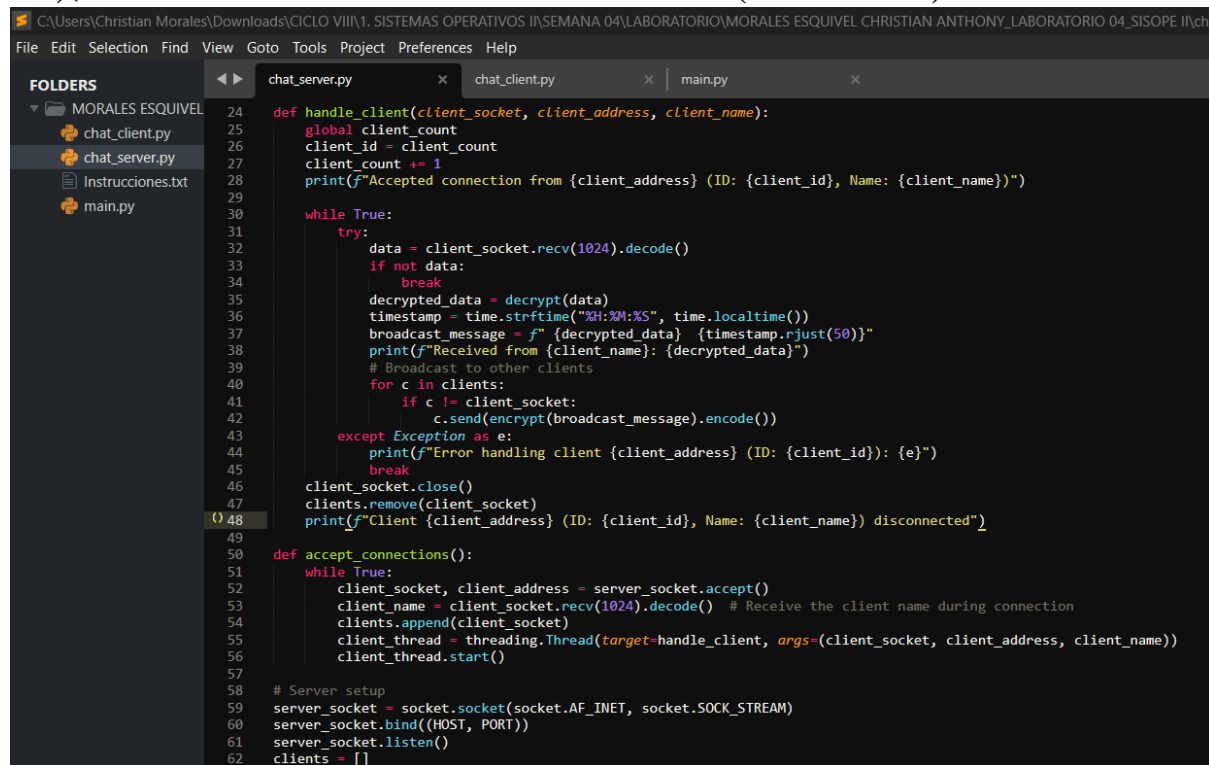
chat_server.py
6  client_count = 0
7
8  HOST = '127.0.0.1'
9  PORT = 65432
10
11 # Simple encryption (for demonstration purposes only -
12 def encrypt(message):
13     encrypted = ""
14     for char in message:
15         encrypted += chr(ord(char) + 1)
16     return encrypted
17
18 def decrypt(message):
19     decrypted = ""
20     for char in message:
21         decrypted += chr(ord(char) - 1)
22     return decrypted
23
```

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 descrypta 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).



```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY_LABORATORIO 04_SISOPE II\chat_server.py
File Edit Selection Find View Goto Tools Project Preferences Help

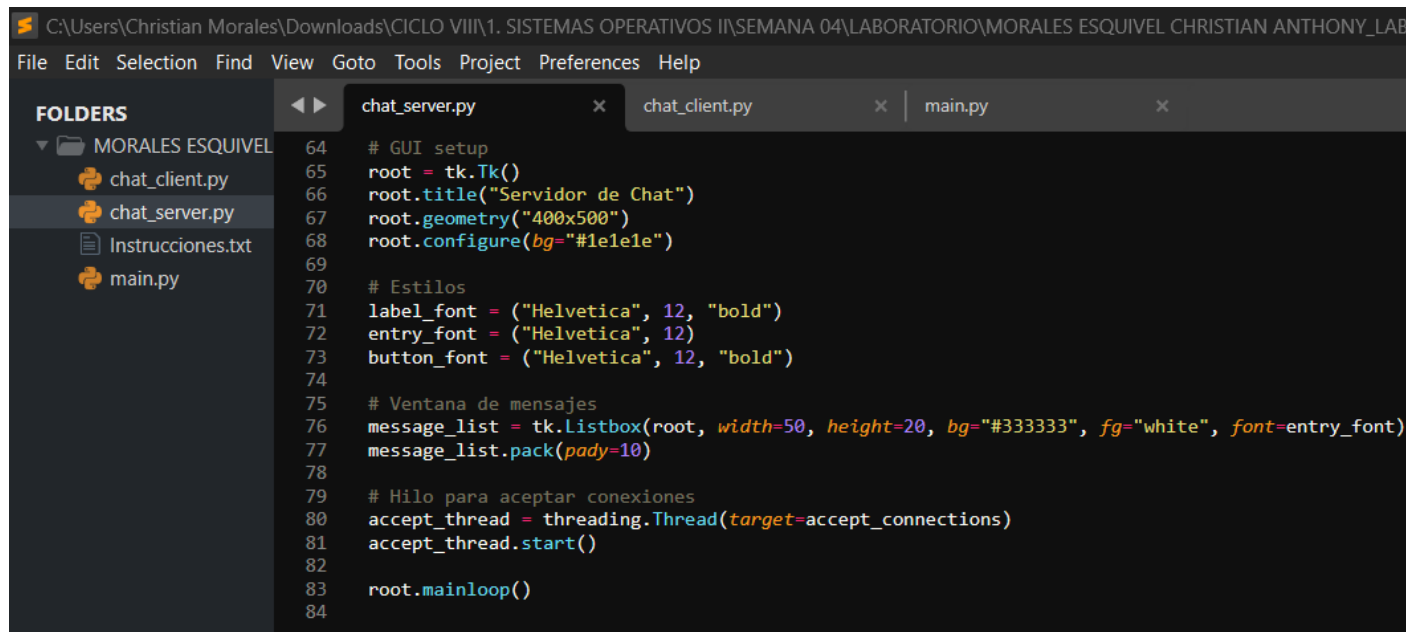
FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py

chat_server.py  x  chat_client.py  x  main.py  x

24 def handle_client(client_socket, client_address, client_name):
25     global client_count
26     client_id = client_count
27     client_count += 1
28     print(f"Accepted connection from {client_address} (ID: {client_id}, Name: {client_name})")
29
30     while True:
31         try:
32             data = client_socket.recv(1024).decode()
33             if not data:
34                 break
35             decrypted_data = decrypt(data)
36             timestamp = time.strftime("%H:%M:%S", time.localtime())
37             broadcast_message = f"{decrypted_data} {timestamp.rjust(50)}"
38             print(f"Received from {client_name}: {decrypted_data}")
39             # Broadcast to other clients
40             for c in clients:
41                 if c != client_socket:
42                     c.send(encrypt(broadcast_message).encode())
43         except Exception as e:
44             print(f"Error handling client {client_address} (ID: {client_id}): {e}")
45             break
46     client_socket.close()
47     clients.remove(client_socket)
48     print(f"Client {client_address} (ID: {client_id}, Name: {client_name}) disconnected")
49
50 def accept_connections():
51     while True:
52         client_socket, client_address = server_socket.accept()
53         client_name = client_socket.recv(1024).decode() # Receive the client name during connection
54         clients.append(client_socket)
55         client_thread = threading.Thread(target=handle_client, args=(client_socket, client_address, client_name))
56         client_thread.start()
57
58 # Server setup
59 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
60 server_socket.bind((HOST, PORT))
61 server_socket.listen()
62 clients = []
```

5. Interfaz Gráfica:

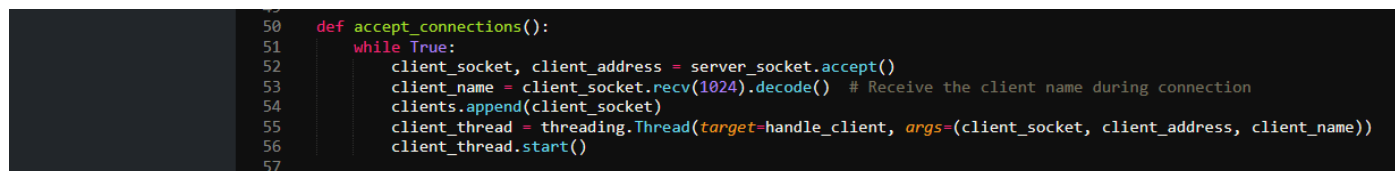
- El servidor cuenta con una sencilla interfaz gráfica creada con Tkinter, que muestra los mensajes en una ventana.
- La ventana principal contiene un Listbox donde se muestra el historial de mensajes.



```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY_LABO
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py
chat_server.py x chat_client.py x main.py x
64 # GUI setup
65 root = tk.Tk()
66 root.title("Servidor de Chat")
67 root.geometry("400x500")
68 root.configure(bg="#1e1e1e")
69
70 # Estilos
71 label_font = ("Helvetica", 12, "bold")
72 entry_font = ("Helvetica", 12)
73 button_font = ("Helvetica", 12, "bold")
74
75 # Ventana de mensajes
76 message_list = tk.Listbox(root, width=50, height=20, bg="#333333", fg="white", font=entry_font)
77 message_list.pack(pady=10)
78
79 # Hilo para aceptar conexiones
80 accept_thread = threading.Thread(target=accept_connections)
81 accept_thread.start()
82
83 root.mainloop()
84
```

6. Aceptación de Conexiones:

- La función `accept_connections` está dedicada a aceptar nuevas conexiones y asignar hilos independientes a cada cliente.



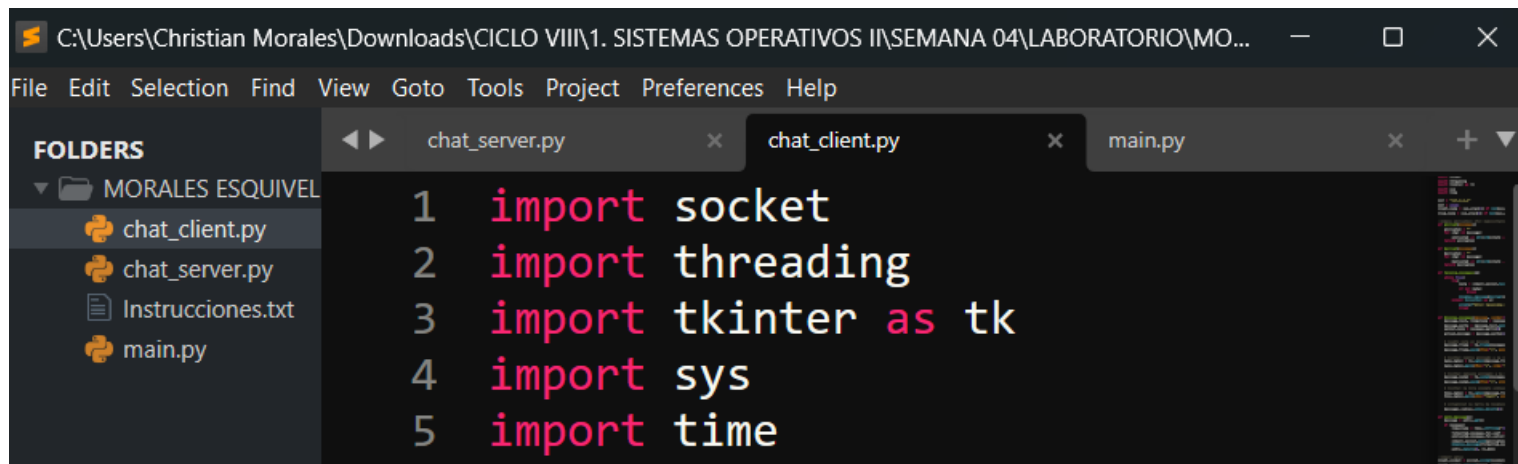
```
50 def accept_connections():
51     while True:
52         client_socket, client_address = server_socket.accept()
53         client_name = client_socket.recv(1024).decode() # Receive the client name during connection
54         clients.append(client_socket)
55         client_thread = threading.Thread(target=handle_client, args=(client_socket, client_address, client_name))
56         client_thread.start()
57
```

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.

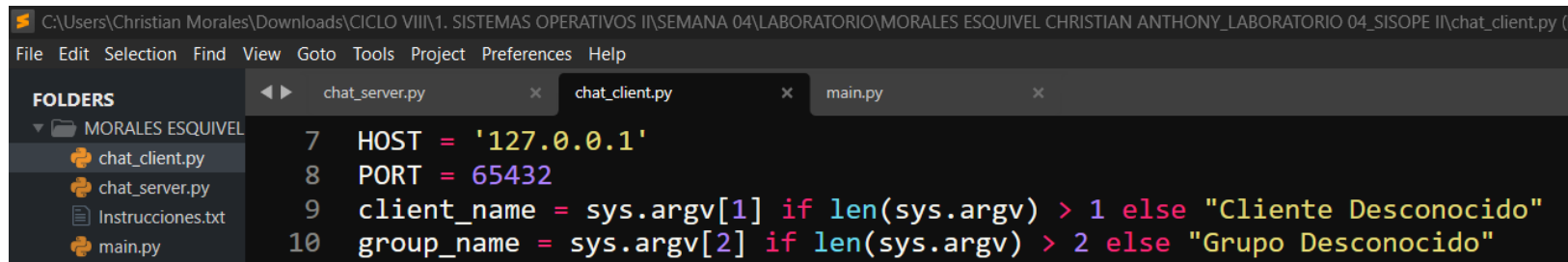


The screenshot shows a code editor window with the title bar 'C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MO...'. The editor has a menu bar with 'File', 'Edit', 'Selection', 'Find', 'View', 'Goto', 'Tools', 'Project', 'Preferences', and 'Help'. On the left, a 'FOLDERS' panel shows a tree structure with 'MORALES ESQUIVEL' as the root, containing 'chat_client.py', 'chat_server.py', 'Instrucciones.txt', and 'main.py'. The 'chat_client.py' file is selected. The main editor area shows the first five lines of code in a dark theme with syntax highlighting: line 1: 'import socket', line 2: 'import threading', line 3: 'import tkinter as tk', line 4: 'import sys', and line 5: 'import time'. The tabs at the top show 'chat_server.py', 'chat_client.py' (active), and 'main.py'.

```
1 import socket
2 import threading
3 import tkinter as tk
4 import sys
5 import time
```


2. Configuración Inicial:

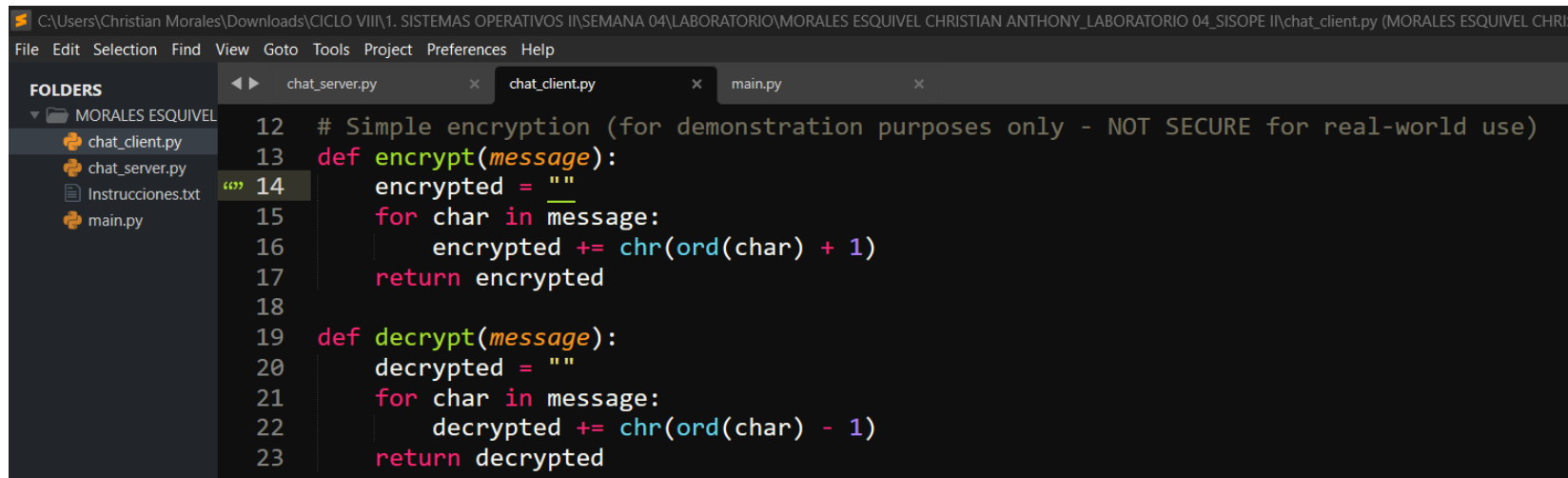
- El cliente se conecta a 127.0.0.1 (localhost) en el puerto 65432.
- El nombre del cliente y el grupo se obtienen de los argumentos de la línea de comandos (sys.argv).



```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY_LABORATORIO 04_SISOPE II\chat_client.py (
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py
7 HOST = '127.0.0.1'
8 PORT = 65432
9 client_name = sys.argv[1] if len(sys.argv) > 1 else "Cliente Desconocido"
10 group_name = sys.argv[2] if len(sys.argv) > 2 else "Grupo Desconocido"
```

3. Funciones de Encriptación:

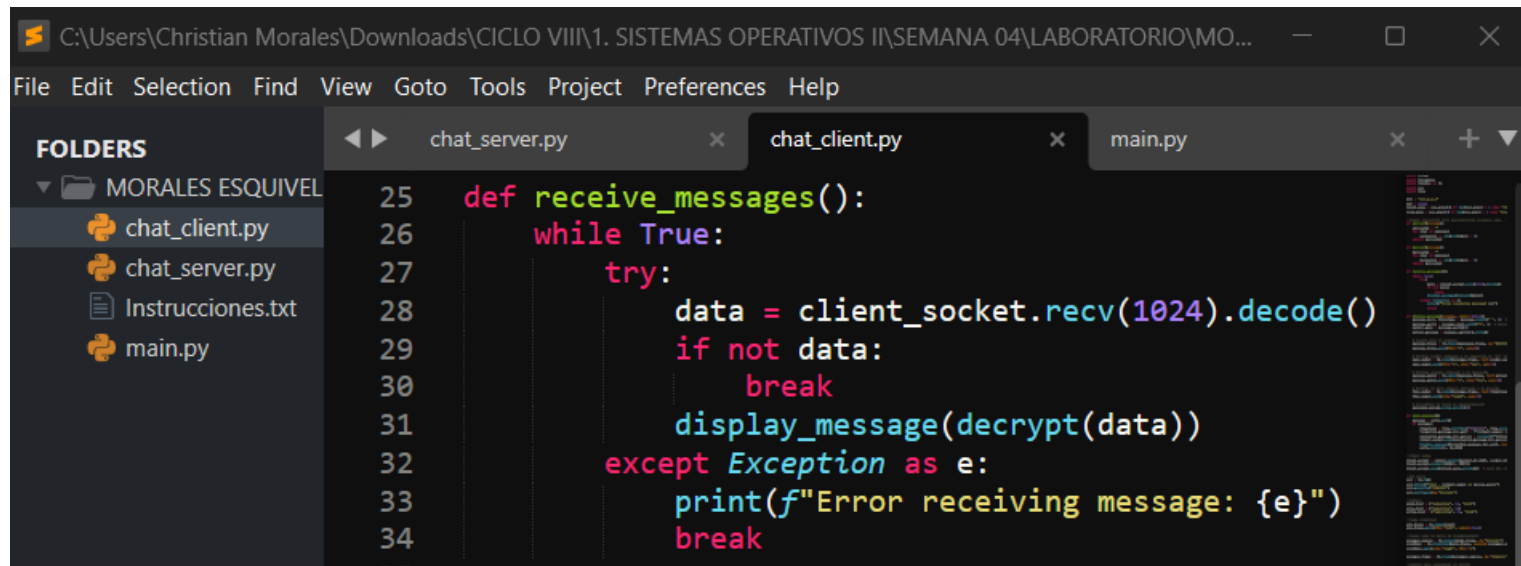
- Similar al servidor, se utilizan las funciones encrypt y decrypt para cifrar y descifrar los mensajes enviados y recibidos.



```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY_LABORATORIO 04_SISOPE II\chat_client.py (MORALES ESQUIVEL CHRI
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py
12 # Simple encryption (for demonstration purposes only - NOT SECURE for real-world use)
13 def encrypt(message):
14     encrypted = ""
15     for char in message:
16         encrypted += chr(ord(char) + 1)
17     return encrypted
18
19 def decrypt(message):
20     decrypted = ""
21     for char in message:
22         decrypted += chr(ord(char) - 1)
23     return decrypted
24
```

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 desenscriptan y se muestran en la ventana del chat.



The screenshot shows a code editor window with the following content:

```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MO...
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
MORALES ESQUIVEL
chat_client.py
chat_server.py
Instrucciones.txt
main.py
25 def receive_messages():
26     while True:
27         try:
28             data = client_socket.recv(1024).decode()
29             if not data:
30                 break
31             display_message(decrypt(data))
32         except Exception as e:
33             print(f"Error receiving message: {e}")
34             break
```

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).

```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY_LABORATORIO 04_SISOPE II\chat_client.py (MORALES ESQUIVEL CHRISTIAN ANTHONY)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py

61 def send_message():
62     message = entry.get()
63     if message:
64         timestamp = time.strftime("%H:%M:%S", time.localtime())
65         formatted_message_for_self = f"{client_name}: {message} {timestamp}" # Formato para el propio cliente
66         encrypted_message_for_server = encrypt(f"{client_name}: {message}") # Solo envía el mensaje
67         client_socket.send(encrypted_message_for_server.encode())
68         display_message(formatted_message_for_self, sender=True) # Marca que es el mensaje del propio cliente
69         entry.delete(0, tk.END)
70
```

6. 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.

```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY_LABORATORIO 04_SISOPE II\chat_client.py (MORALES ESQUIVEL CHRISTIAN ANTHONY)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py

35
36 def display_message(message, sender=False):
37     message_text, timestamp = message.rsplit(' ', 1) # Separa el mensaje del timestamp
38     message_parts = message_text.split(':', 1) # Separar nombre del mensaje
39     sender_name = message_parts[0]
40     actual_message = message_parts[1].strip()
41
42     # Cuadro para el mensaje
43     message_frame = tk.Frame(messages_frame, bg="#333333")
44     message_frame.pack(fill='x', pady=5)
45
46     # Mostrar nombre alineado a la izquierda (o "Yo" si es el mensaje del propio cliente)
47     name_label = tk.Label(message_frame, text=sender_name if not sender else "Yo", bg="#333333", fg="#4caf50", font=("Helvetica", 10, "bold"), anchor="w")
48     name_label.pack(fill="x", side="top", padx=5)
49
50     # Mostrar mensaje alineado a la izquierda
51     message_label = tk.Label(message_frame, text=actual_message, wraplength=300, bg="#333333", fg="white", font=entry_font, justify="left", anchor="w")
52     message_label.pack(fill="x", side="top", padx=5)
53
54     # Mostrar la hora siempre alineada a la derecha
55     time_label = tk.Label(message_frame, text=timestamp, bg="#333333", fg="white", font=("Helvetica", 10))
56     time_label.pack(side='right', padx=5)
57
58     # Actualizar la barra de desplazamiento
59     messages_canvas.yview_moveto(1)
60
61 def send_message():
62     message = entry.get()
63     if message:
64         timestamp = time.strftime("%H:%M:%S", time.localtime())
65         formatted_message_for_self = f"{client_name}: {message} {timestamp}" # Formato para el propio cliente
66         encrypted_message_for_server = encrypt(f"{client_name}: {message}") # Solo envía el mensaje
67         client_socket.send(encrypted_message_for_server.encode())
68         display_message(formatted_message_for_self, sender=True) # Marca que es el mensaje del propio cliente
69         entry.delete(0, tk.END)
70
```

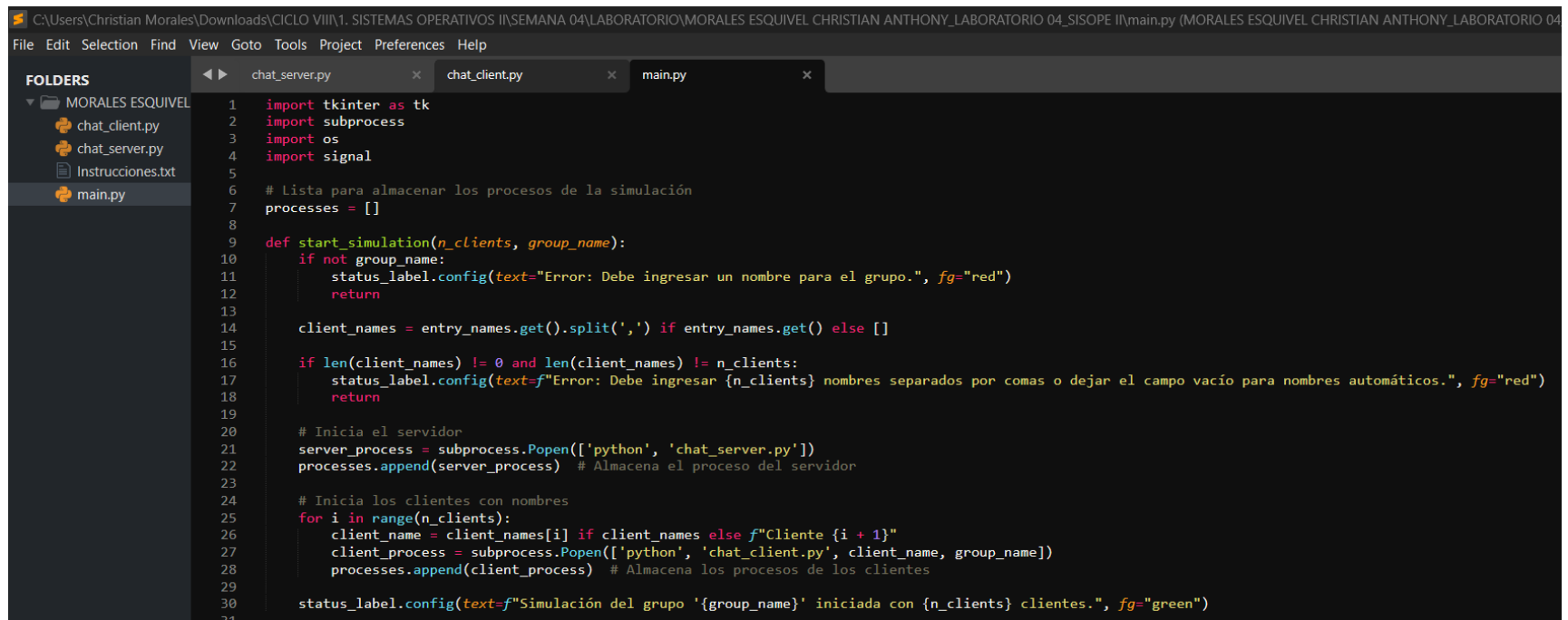
```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY_LABORATORIO 04_SISOPE II\chat_client.py (MORALES ESQ
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
MORALES ESQUIVEL
chat_client.py
chat_server.py
Instrucciones.txt
main.py
70
71 # Client setup
72 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
73 client_socket.connect((HOST, PORT))
74 client_socket.send(client_name.encode()) # Send the client name to the server
75
76 # GUI setup
77 root = tk.Tk()
78 root.title(f"Chat - {client_name} en {group_name}")
79 root.geometry("400x500")
80 root.configure(bg="#1e1e1e")
81
82 # Estilos
83 label_font = ("Helvetica", 12, "bold")
84 entry_font = ("Helvetica", 12)
85 button_font = ("Helvetica", 12, "bold")
86
87 # Frame principal
88 main_frame = tk.Frame(root)
89 main_frame.pack(fill='both', expand=True)
90
91 # Canvas para la barra de desplazamiento
92 messages_canvas = tk.Canvas(main_frame, bg="#1e1e1e")
93 scrollbar = tk.Scrollbar(main_frame, command=messages_canvas.yview)
94 scrollbar.pack(side="right", fill="y")
95
96 messages_frame = tk.Frame(messages_canvas, bg="#1e1e1e")
97
98 # Función para actualizar el scroll
99 def on_configure(event):
100     messages_canvas.configure(scrollregion=messages_canvas.bbox("all"))
101
102 messages_frame.bind("<Configure>", on_configure)
103 messages_canvas.create_window((0, 0), window=messages_frame, anchor="nw")
104 messages_canvas.pack(side="left", fill="both", expand=True)
105 messages_canvas.configure(yscrollcommand=scrollbar.set)
106
107 # Campo de entrada para mensajes
108 entry = tk.Entry(root, font=entry_font, width=30, bg="#2e2e2e", fg="white")
109 entry.pack(pady=10)
110
111 # Botón de enviar
112 send_button = tk.Button(root, text="Enviar", font=button_font, bg="#4caf50", fg="white", activebackground="#45a049", width=15, command=send_message)
113 send_button.pack(pady=10)
114
115 # Hilo para recibir mensajes
116 receive_thread = threading.Thread(target=receive_messages)
117 receive_thread.start()
118
119 root.mainloop()
120
```

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.



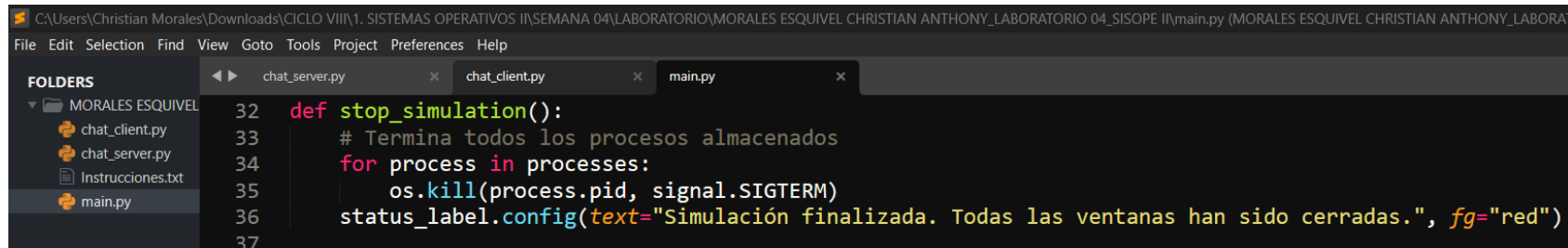
```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY_LABORATORIO 04_SISOPE II\main.py (MORALES ESQUIVEL CHRISTIAN ANTHONY_LABORATORIO 04_SISOPE II)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
▼ MORALES ESQUIVEL
  chat_client.py
  chat_server.py
  Instrucciones.txt
  main.py

1 import tkinter as tk
2 import subprocess
3 import os
4 import signal
5
6 # Lista para almacenar los procesos de la simulación
7 processes = []
8
9 def start_simulation(n_clients, group_name):
10     if not group_name:
11         status_label.config(text="Error: Debe ingresar un nombre para el grupo.", fg="red")
12         return
13
14     client_names = entry_names.get().split(',') if entry_names.get() else []
15
16     if len(client_names) != 0 and len(client_names) != n_clients:
17         status_label.config(text=f"Error: Debe ingresar {n_clients} nombres separados por comas o dejar el campo vacío para nombres automáticos.", fg="red")
18         return
19
20     # Inicia el servidor
21     server_process = subprocess.Popen(['python', 'chat_server.py'])
22     processes.append(server_process) # Almacena el proceso del servidor
23
24     # Inicia los clientes con nombres
25     for i in range(n_clients):
26         client_name = client_names[i] if client_names else f"Cliente {i + 1}"
27         client_process = subprocess.Popen(['python', 'chat_client.py', client_name, group_name])
28         processes.append(client_process) # Almacena los procesos de los clientes
29
30     status_label.config(text=f"Simulación del grupo '{group_name}' iniciada con {n_clients} clientes.", fg="green")
31
```

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.



The screenshot shows a code editor with a dark theme. The left sidebar displays a file explorer with the following structure:

- MORALES ESQUIVEL
 - chat_client.py
 - chat_server.py
 - Instrucciones.txt
 - main.py

The main editor area shows the `main.py` file with the following code:

```
32 def stop_simulation():
33     # Termina todos los procesos almacenados
34     for process in processes:
35         os.kill(process.pid, signal.SIGTERM)
36     status_label.config(text="Simulación finalizada. Todas las ventanas han sido cerradas.", fg="red")
37
```

3. 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`).

4. Validación:

- Valida la entrada del número de clientes y asegura que el nombre del grupo esté presente.

```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY LABORATORIO 04_SISOPE II\main.py (MORALES ESQUIVEL CHRISTIAN ANTHONY LABORATORIO 04_SISOPE II)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py

38 def request_client_names():
39     try:
40         n_clients = int(entry_clients.get())
41         if n_clients < 1:
42             raise ValueError("El número de clientes debe ser al menos 1")
43
44         # Limpiar y mostrar el campo para nombres y el botón de simulación
45         for widget in root.winfo_children():
46             widget.destroy()
47
48         instruction_group = tk.Label(root, text="Ingrese el nombre del grupo:", font=label_font, bg="#1e1e1e", fg="white")
49         instruction_group.pack(pady=10)
50
51         global entry_group_name
52         entry_group_name = tk.Entry(root, font=entry_font, width=30, justify="center")
53         entry_group_name.pack(pady=10)
54
55         instruction_names = tk.Label(root, text=f"Ingrese los nombres de los {n_clients} clientes (opcional):", font=label_font, bg="#1e1e1e", fg="white")
56         instruction_names.pack(pady=10)
57
58         global entry_names
59         entry_names = tk.Entry(root, font=entry_font, width=30, justify="center")
60         entry_names.pack(pady=10)
61
62         start_button = tk.Button(root, text="Iniciar Simulación", font=button_font, bg="#4caf50", fg="white", activebackground="#45a049", width=20,
63                                 command=lambda: start_simulation(n_clients, entry_group_name.get()))
64         start_button.pack(pady=20)
65
66         stop_button = tk.Button(root, text="Finalizar Simulación", font=button_font, bg="#f44336", fg="white", activebackground="#e53935", width=20, command=stop_simulation)
67         stop_button.pack(pady=20)
68
69         global status_label
70         status_label = tk.Label(root, text="", font=label_font, bg="#1e1e1e", fg="white", wraplength=350)
71         status_label.pack(pady=10)
72
73     except ValueError as e:
74         status_label.config(text=f"Error: {e}", fg="red")
75
76 # Configuración de la ventana principal
```

```
C:\Users\Christian Morales\Downloads\CICLO VIII\1. SISTEMAS OPERATIVOS II\SEMANA 04\LABORATORIO\MORALES ESQUIVEL CHRISTIAN ANTHONY LABORATORIO 04_SISOPE II\main.py (MORALES ESQUIVEL CHRISTIAN ANTHONY LABORATORIO 04_SISOPE II)
File Edit Selection Find View Goto Tools Project Preferences Help

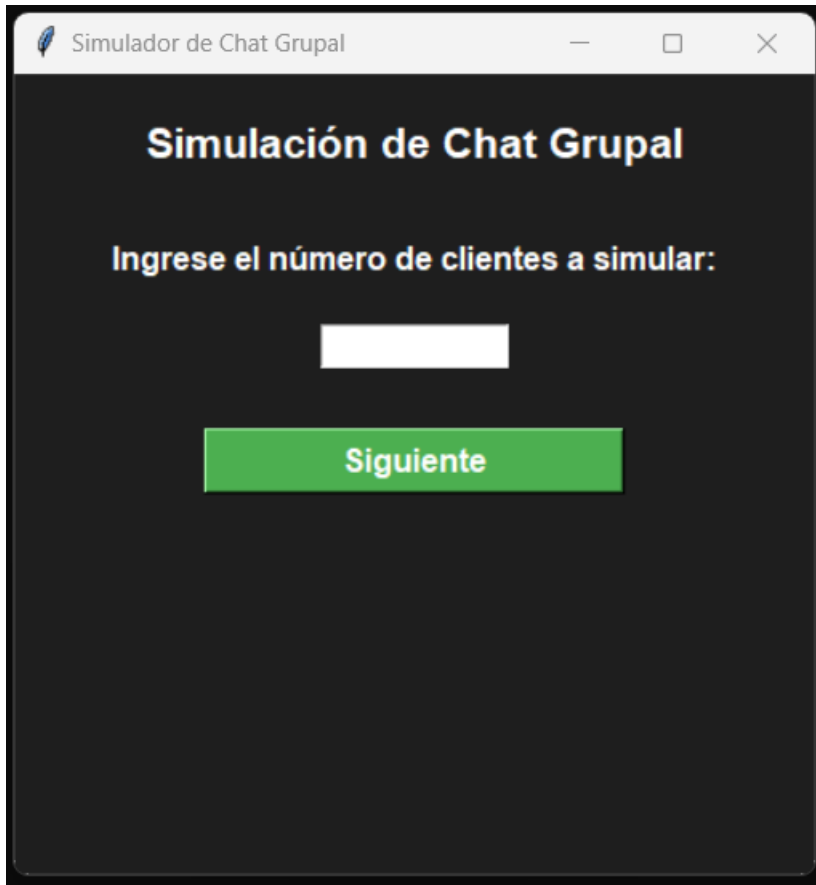
FOLDERS
  MORALES ESQUIVEL
    chat_client.py
    chat_server.py
    Instrucciones.txt
    main.py

76 # Configuración de la ventana principal
77 root = tk.Tk()
78 root.title("Simulador de Chat Grupal")
79 root.geometry("400x400")
80 root.configure(bg="#1e1e1e") # Fondo oscuro
81
82 # Estilos
83 label_font = ("Helvetica", 12, "bold")
84 entry_font = ("Helvetica", 12)
85 button_font = ("Helvetica", 12, "bold")
86
87 # Título de la aplicación
88 title_label = tk.Label(root, text="Simulación de Chat Grupal", font=("Helvetica", 16, "bold"), bg="#1e1e1e", fg="white")
89 title_label.pack(pady=20)
90
91 # Instrucciones para el número de clientes
92 instruction_label = tk.Label(root, text="Ingrese el número de clientes a simular:", font=label_font, bg="#1e1e1e", fg="white")
93 instruction_label.pack(pady=10)
94
95 # Campo de entrada para el número de clientes
96 entry_clients = tk.Entry(root, font=entry_font, width=10, justify="center")
97 entry_clients.pack(pady=10)
98
99 # Botón para continuar a la solicitud de nombres
100 next_button = tk.Button(root, text="Siguiente", font=button_font, bg="#4caf50", fg="white", activebackground="#45a049", width=20, command=request_client_names)
101 next_button.pack(pady=20)
102
103 # Etiqueta para mostrar mensajes de estado
104 status_label = tk.Label(root, text="", font=label_font, bg="#1e1e1e", fg="white", wraplength=350)
105 status_label.pack(pady=10)
106
107 root.mainloop()
108
```

Ejecución del Código:

Ventana Principal

Vista Principal Parte I



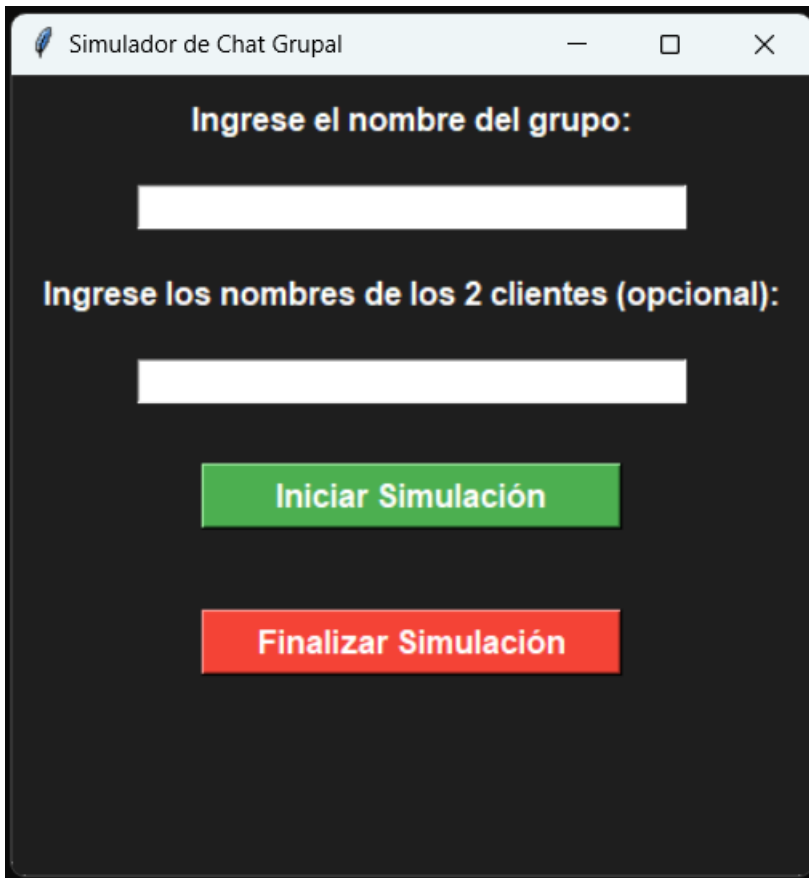
Simulador de Chat Grupal

Simulación de Chat Grupal

Ingrese el número de clientes a simular:

Siguiente

Vista Principal Parte II



Simulador de Chat Grupal

Ingrese el nombre del grupo:

Ingrese los nombres de los 2 clientes (opcional):

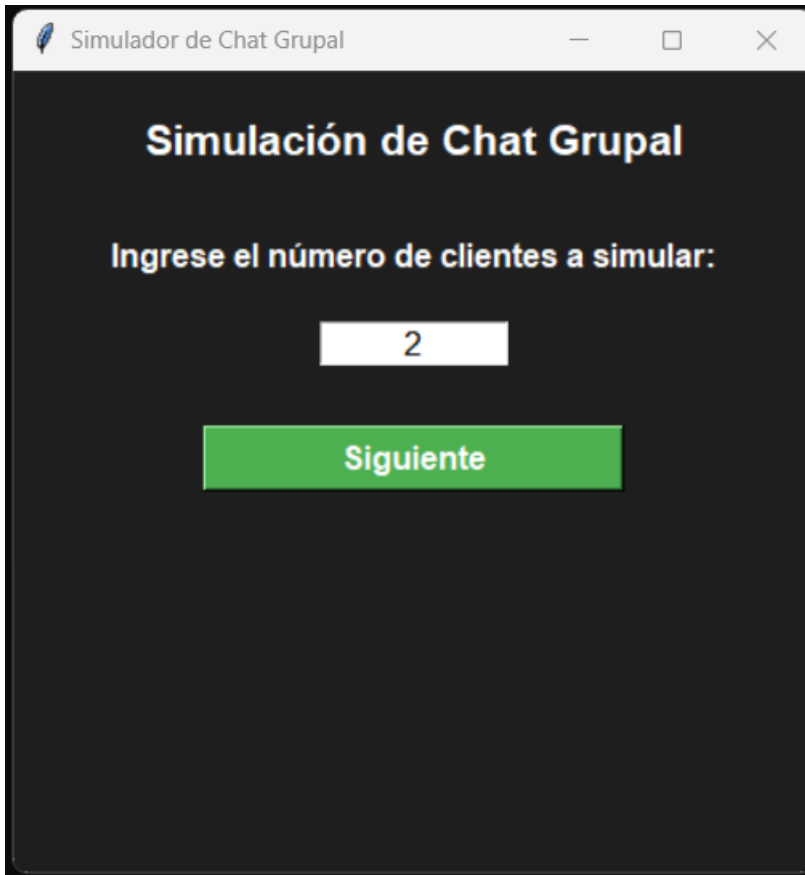
Iniciar Simulación

Finalizar Simulación

Ejecución del Código:

Ventana Principal

Vista Principal Parte I



Simulador de Chat Grupal

Simulación de Chat Grupal

Ingrese el número de clientes a simular:

2

Siguiente

Vista Principal Parte II



Simulador de Chat Grupal

Ingrese el nombre del grupo:

Pelotero FC

Ingrese los nombres de los 2 clientes (opcional):

David, José

Iniciar Simulación

Finalizar Simulación

Simulación de Chat Grupal

The screenshot displays a Windows desktop environment with a dark theme. At the top, a terminal window titled 'C:\WINDOWS\system32\cmd.' shows the following log:

```
Accepted connection from ('127.0.0.1', 55811) (ID: 0, Name: David)
Accepted connection from ('127.0.0.1', 55812) (ID: 1, Name: José)
Received from David: David: Hola
Received from José: José: Hola, que tal?
Received from David: David: Muy bien gracias y tu?
Received from José: José: Excelentee!
Received from David: David: Y el resto de chicos?
Received from David: David: Ya no salen a jugar o k?
Received from José: José: Siii, pero solo los fines de semana porque la mayoría estudia pues
Received from José: José: :/
```

Below the terminal are three application windows:

- Servidor de Chat:** An empty chat window with a text input field and an 'Enviar' button.
- Chat - David en Pelotero FC:** A chat window showing messages from 'Yo' (David) and 'José'. The messages are: 'Hola' (06:16:54), 'Hola, que tal?' (06:17:04), 'Muy bien gracias y tu?' (06:17:15), 'Excelentee!' (06:17:26), and 'Y el resto de chicos?' (06:17:42). It includes a text input field and an 'Enviar' button.
- Chat - José en Pelotero FC:** A chat window showing messages from 'Yo' (David) and 'Yo' (José). The messages are: 'Excelentee!' (06:17:26), 'Y el resto de chicos?' (06:17:42), 'Ya no salen a jugar o k?' (06:17:49), 'Siii, pero solo los fines de semana porque la mayoría estudia pues' (06:18:14), and ':/' (06:18:22). It includes a text input field and an 'Enviar' button.

On the right side of the desktop is a window titled 'Simulador de Chat Grupal' with the following controls:

- Ingrese el nombre del grupo:** A text input field containing 'Pelotero FC'.
- Ingrese los nombres de los 2 clientes (opcional):** A text input field containing 'David,José'.
- Iniciar Simulación:** A green button.
- Finalizar Simulación:** A red button.
- Simulación del grupo 'Pelotero FC' iniciada con 2 clientes.** A green status message.

The Windows taskbar at the bottom shows various application icons and the system clock indicating 06:20 a. m. on 9/10/2024.