UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

Facultad de Ingeniería

UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

Ingeniería en Ciencias de la Computación

Cómputo paralelo y distribuido

Proyecto primer parcial

Trabajo de:

Manuel Abraham Escudero Moreno	355208
Adrian (Adora)González Domínguez	359834
Héctor Daniel Medrano Meza	361345

Asesor: De Lira Miramontes Jose Saul

15 de octubre de 2024

Proyecto I

Desarrollar una aplicación de chat en Python que permita la comunicación en tiempo real entre múltiples usuarios. La aplicación debe aprovechar el procesamiento en paralelo mediante el uso de hilos (threads) o procesos, para manejar eficientemente múltiples conexiones y mensajes simultáneos.

Requisitos:

1. Servidor de Chat
    a. Debe aceptar conexiones entrantes de múltiples clientes simultáneamente.
    b. Gestionará el envío y recepción de mensajes entre los clientes conectados.
    c. Distribuirá los mensajes entrantes al destinatario correcto o a todos los usuarios en el caso de mensajes de difusión.
2. Cliente de Chat
    a. Permitirá a los usuarios conectarse al servidor y enviar/recibir mensajes en tiempo real.
    b. Tendrá una interfaz para ingresar mensajes y visualizar las conversaciones.
3. Procesamiento en Paralelo
    a. Utilizar hilos (módulo `threading`) o procesos (módulo `multiprocessing`) para manejar cada conexión de cliente de forma independiente.
    b. Asegurar que el servidor pueda atender nuevas conexiones y mensajes sin bloquearse.
4. Comunicación en Red
    a. Implementar protocolos de comunicación utilizando sockets
    b. Utilizar protocolos TCP para conexiones fiables.
5. Sincronización y Seguridad
    a. Manejar adecuadamente la sincronización entre hilos o procesos para evitar race condition
    b. Utilizar mecanismos de bloqueo (locks) cuando sea necesario.

6. Escalabilidad y Rendimiento
    a. Optimizar el uso de recursos del sistema para manejar un gran número de conexiones simultáneas.
    b. Evaluar el rendimiento y ajustar la arquitectura (hilos vs. procesos) según las necesidades.

Consideraciones Adicionales

- Manejo de Errores: Implementar manejo de excepciones para conexiones interrumpidas y otros errores de E/S.
- Extensibilidad: Diseñar el sistema de manera modular para facilitar futuras ampliaciones, como salas de chat privadas, autenticación de usuarios, o transferencia de archivos.
- Interfaz de Usuario Gráfica

Codigo del servidor

```
from dotenv import load_dotenv
import os
import uvicorn
import threading
import asyncio
import logging
from threading import Semaphore, Thread, Lock
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
import json

import signal
signal.signal(signal.SIGINT, signal.SIG_DFL)

logging.basicConfig(level=logging.DEBUG,
format='%(threadName)s: %(message)s')

maximum_client_count: int = 4
ws_app = FastAPI()

class ConnectionManager:
```

```
    def __init__(self):
        self.active_connections = []
        self.lock=Lock()
        self.semaphore = Semaphore(maximum_client_count)

    async def connect(self, websocket: WebSocket,
client_id:str):
        if self.semaphore.acquire(blocking=False):
            with self.lock:
                await websocket.accept()

self.active_connections.append([websocket,client_id])
                return True
        else:
            # Manejar el caso cuando no hay espacios
disponibles
            logging.warning("No hay espacios disponibles en el
semáforo.")
            return False

    async def disconnect(self, websocket: WebSocket,
client_id:str):
        self.semaphore.release()
        with self.lock:

self.active_connections.remove([websocket,client_id])

    async def send_personal_message(self, message: str,
websocket: WebSocket):
        await websocket.send_text(message)

    async def broadcast(self, message: str, ws: WebSocket =
None):
        for connection in self.active_connections:
            if ws != connection[0]:
                try:
                    await connection[0].send_text(message)
```

```
                except Exception as e:
                    logging.error(f"Error sending message:
{e}")

    def get_ws(self, client_id: str):
        for connection in self.active_connections:
            print(client_id, connection[1])
            if client_id.strip() == connection[1].strip():
                return connection[0]
        return False



manager = ConnectionManager()
exceptions_manager = ConnectionManager()

@ws_app.get("/")
def read_root():
    return {"content": "Hello World"}

@ws_app.websocket("/ws/{client_id}")
async def websocket_endpoint(websocket: WebSocket, client_id:
str):
    logging.info(f"Trying to connect {client_id}")
    try:
        has_connected=await
manager.connect(websocket,client_id)
        if(has_connected):
            logging.info("CONECTION STABLISHED")
            while True:
                text = await websocket.receive_text()
                parse=text.split(',')
                data=[','.join(parse[:len(parse)-1]),
parse[-1]]
                logging.info(parse)
                logging.info(data)
```

```
                Thread(target=handle_message, args=(websocket,
client_id, data)).start()
        else:
            await notify_problem(websocket,client_id)

    except WebSocketDisconnect:
        await manager.disconnect(websocket, client_id)
        await manager.broadcast(f"Client #{client_id} left the
chat", websocket)

async def notify_problem(websocket: WebSocket, client_id:
int):
    await exceptions_manager.connect(websocket,client_id)

    await exceptions_manager.send_personal_message("Server
full", websocket)
    await exceptions_manager.disconnect(websocket,client_id)

def handle_message(websocket: WebSocket, client_id: int, data:
any):
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    loop.run_until_complete(process_message(websocket,
client_id, data))
    logging.info("Send message completed")

async def process_message(websocket: WebSocket, client_id:
int, data: any):
    await manager.send_personal_message(f"You wrote:
{data[0]}", websocket)
    if(data[1]):
        ws = manager.get_ws(data[1])
        if(ws):
            print(type(ws))
            await manager.send_personal_message(f"{client_id}
send a private message: {data[0]}", ws)
        else:
```

```
            await manager.send_personal_message(f"{data[1]} is
not a valid client", websocket)
    else:
        await manager.broadcast(f"Client #{client_id} says:
{data[0]}", websocket)

def run():
    ENVIRONMENT = os.getenv("ENVIRONMENT", "development")  #
Por defecto a desarrollo

    if ENVIRONMENT == "production":
        load_dotenv(dotenv_path='.env.prod')
    else:
        load_dotenv(dotenv_path='.env')

    host = os.getenv("HOST")
    port = int(os.getenv("PORT"))

    config = uvicorn.Config(ws_app, host=host, port=port,
log_level="info", workers=4)
    server = uvicorn.Server(config)
    server.run()

if __name__=="__main__":

    run()
```

En el servidor por mas que intentamos, dado que es un servidor de tipo ASGI (por FastAPI). No logramos establecer que cada conexión se manejase en un hilo por separado. ASGI no esta diseñado para soportar multiples hilo. Sin embargo, uvicorn se puede configurar para trabajar en varios procesos paralelos, configurando la opción de workers se puede indicar en cuantos procesos en paralelo funcionará el código.

Codigo cliente

```
from dotenv import load_dotenv
import os
import uvicorn
from fastapi import FastAPI
from fastapi.staticfiles import StaticFiles
```

```
from python_event_bus import EventBus

host=""
serverName=""
port=0

app = FastAPI()

@app.get("/env")
async def get_env_variables():
    return {
        "HOST": host,
        "PORT": port,
        "SERVER": serverName
    }

app.mount('/', StaticFiles(directory='static', html=True),
name='static')


def run():
    global host,port,serverName

    ENVIRONMENT = os.getenv("ENVIRONMENT", "development")  #
Por defecto a desarrollo

    if ENVIRONMENT == "production":
        load_dotenv(dotenv_path='.env.prod')
    else:
        load_dotenv(dotenv_path='.env')

    host = os.getenv("HOST", "0.0.0.0")
    port = int(os.getenv("PORT", 8080))
    serverName = os.getenv("SERVER", "localhost:8080")

    config = uvicorn.Config(app, host=host, port=port,
log_level="info")
```

```
    server = uvicorn.Server(config)
    server.run()

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

Enlaces
Repositorio: https://github.com/stariluz/uach-pdc-chat
Chat: https://uachpdcchatclient-pk5t9bsa.b4a.run/
Server: https://uachpdcchatserver-cbeg6f8n.b4a.run/