import socket

import threading

import json

import random

import time

# Server Configuration

HOST = '0.0.0.0'  # Accept connections from any IP

PORT = 5000

jugadores = []

colores\_disponibles = ["red", "blue", "green", "yellow"]

turno\_actual\_idx = 0

bloqueo\_turnos = threading.Lock()

max\_jugadores = 4

min\_jugadores\_para\_iniciar = 2

juego\_iniciado = False

dados\_lanzados = False

ultimo\_dado\_resultado = 0

ultimo\_dado\_dobles = False

# Definiciones de caminos y casillas especiales

CAMINO\_GLOBAL = [

    (7,0),(7,1),(7,2),(7,3),(7,4),(7,5),(7,6),

    (6,6),(5,6),(4,6),(3,6),(2,6),(1,6),(0,6),

    (0,7),(0,8),(0,9),(0,10),(0,11),

    (1,11),(2,11),(3,11),(4,11),(5,11),(6,11),

    (7,11),(7,12),(7,13),(7,14),(7,15),(7,16),(7,17),

    (8,17),(9,17),(10,17),(11,17),

    (11,16),(11,15),(11,14),(11,13),(11,12),(11,11),

    (11,10),(12,10),(13,10),(14,10),(15,10),(16,10),(17,10),

    (17,9),(17,8),(17,7),(17,6),

    (16,6),(15,6),(14,6),(13,6),(12,6),(11,6),

    (10,6),(10,5),(10,4),(10,3),(10,2),(10,1),(10,0),

    (9,0),(8,0),(7,0)

]

# Puntos de entrada al camino global desde las cárceles

ENTRADAS\_GLOBAL = {

    "red": 0,

    "blue": 17,

    "green": 34,

    "yellow": 51

}

# Caminos de meta (casa) para cada color

CAMINOS\_META = {

    "red": [(8,1), (8,2), (8,3), (8,4), (8,5), (8,6), (8,7), (8,8)],

    "blue": [(1,8), (2,8), (3,8), (4,8), (5,8), (6,8), (7,8), (8,8)],

    "green": [(9,16), (9,15), (9,14), (9,13), (9,12), (9,11), (9,10), (8,8)],

    "yellow": [(16,9), (15,9), (14,9), (13,9), (12,9), (11,9), (10,9), (8,8)]

}

# Casillas seguras

CASILLAS\_SEGURAS = [

    (7,0), (0,6), (0,11), (7,11), (7,17), (11,17), (11,10), (17,10),

    (17,6), (11,6), (10,0), (4,6), (7,13), (11,4), (13,10)

]

# Coordenadas de las cárceles

CARCELES\_COORDS = {

    "red": [(1, 1), (1, 2), (2, 1), (2, 2)],

    "blue": [(1, 15), (1, 16), (2, 15), (2, 16)],

    "green": [(15, 1), (15, 2), (16, 1), (16, 2)],

    "yellow": [(15, 15), (15, 16), (16, 15), (16, 16)]

}

class Jugador:

    def \_\_init\_\_(self, conn, addr, nombre, color):

        self.conn = conn

        self.addr = addr

        self.nombre = nombre

        self.color = color

        # Estados: 0=cárcel, 1=camino\_global, 2=camino\_meta, 3=meta\_final

        self.ficha\_estado = [0] \* 4

        # Posiciones: -1=cárcel, índice en camino correspondiente

        self.ficha\_pos = [-1] \* 4

        self.ultimo\_dado = 0

        self.pares\_consecutivos = 0

        self.fichas\_en\_meta\_final = [False] \* 4

        self.puede\_relanzar = False

def enviar\_mensaje(jugador\_o\_conn, mensaje\_dict):

    """Envía un mensaje JSON a un jugador o conexión"""

    try:

        mensaje\_str = json.dumps(mensaje\_dict)

        if isinstance(jugador\_o\_conn, Jugador):

            jugador\_o\_conn.conn.send(mensaje\_str.encode())

        else:

            jugador\_o\_conn.send(mensaje\_str.encode())

        return True

    except Exception as e:

        print(f"[-] Error al enviar mensaje: {e}")

        return False

def enviar\_a\_todos(mensaje\_dict, except\_jugador=None):

    """Envía un mensaje a todos los jugadores excepto uno específico"""

    for jugador in jugadores[:]:  # Copia para evitar problemas de concurrencia

        if jugador != except\_jugador:

            if not enviar\_mensaje(jugador, mensaje\_dict):

                # Si falla el envío, el jugador probablemente se desconectó

                print(f"[-] Falló envío a {jugador.nombre}, posible desconexión")

def obtener\_coordenadas\_ficha(jugador, ficha\_idx):

    """Obtiene las coordenadas actuales de una ficha"""

    estado = jugador.ficha\_estado[ficha\_idx]

    pos = jugador.ficha\_pos[ficha\_idx]

    if estado == 0:  # En cárcel

        return CARCELES\_COORDS[jugador.color][ficha\_idx]

    elif estado == 1:  # En camino global

        return CAMINO\_GLOBAL[pos]

    elif estado == 2 or estado == 3:  # En camino meta o meta final

        return CAMINOS\_META[jugador.color][pos]

    return None

def es\_casilla\_segura(coords):

    """Verifica si una casilla es segura"""

    return coords in CASILLAS\_SEGURAS

def obtener\_posibles\_movimientos(jugador, dado\_total):

    """Obtiene las fichas que se pueden mover con el dado actual"""

    movible\_fichas = []

    # Si sacó pares, puede sacar fichas de la cárcel

    if ultimo\_dado\_dobles:

        for i in range(4):

            if jugador.ficha\_estado[i] == 0:  # En cárcel

                movible\_fichas.append(i)

        if movible\_fichas:  # Si hay fichas en cárcel, solo puede sacar

            return movible\_fichas

    # Verificar fichas que se pueden mover

    for i in range(4):

        estado = jugador.ficha\_estado[i]

        pos = jugador.ficha\_pos[i]

        if estado == 1:  # En camino global

            nueva\_pos = pos + dado\_total

            # Verificar si entra a la meta

            entrada\_meta = ENTRADAS\_GLOBAL[jugador.color] + len(CAMINO\_GLOBAL) - 7

            if nueva\_pos >= entrada\_meta:

                pasos\_en\_meta = nueva\_pos - entrada\_meta

                if pasos\_en\_meta < len(CAMINOS\_META[jugador.color]):

                    movible\_fichas.append(i)

            elif nueva\_pos < len(CAMINO\_GLOBAL):

                movible\_fichas.append(i)

        elif estado == 2:  # En camino meta

            nueva\_pos\_meta = pos + dado\_total

            if nueva\_pos\_meta < len(CAMINOS\_META[jugador.color]):

                movible\_fichas.append(i)

    return movible\_fichas

def verificar\_captura(jugador\_atacante, coords\_destino, ficha\_atacante):

    """Verifica si se puede capturar una ficha enemiga"""

    if es\_casilla\_segura(coords\_destino):

        return None

    for otro\_jugador in jugadores:

        if otro\_jugador == jugador\_atacante:

            continue

        for i in range(4):

            if otro\_jugador.ficha\_estado[i] == 1:  # Solo fichas en camino global

                coords\_otra = obtener\_coordenadas\_ficha(otro\_jugador, i)

                if coords\_otra == coords\_destino:

                    return (otro\_jugador, i)

    return None

def verificar\_victoria(jugador):

    """Verifica si un jugador ha ganado"""

    return all(jugador.fichas\_en\_meta\_final)

def enviar\_turno():

    """Envía notificación de turno a todos los jugadores"""

    global turno\_actual\_idx, dados\_lanzados

    with bloqueo\_turnos:

        if not jugadores:

            return

        jugador\_actual = jugadores[turno\_actual\_idx]

        dados\_lanzados = False

        jugador\_actual.puede\_relanzar = False

        print(f"[+] Turno de {jugador\_actual.nombre} ({jugador\_actual.color})")

        for i, jugador in enumerate(jugadores):

            es\_tu\_turno = (i == turno\_actual\_idx)

            mensaje = {

                "tipo": "turno",

                "mensaje": f"Es tu turno, {jugador.nombre}." if es\_tu\_turno else f"Turno de {jugador\_actual.nombre}.",

                "es\_tu\_turno": es\_tu\_turno,

                "puede\_lanzar": es\_tu\_turno

            }

            enviar\_mensaje(jugador, mensaje)

def manejar\_lanzamiento\_dado(jugador):

    """Maneja el lanzamiento de dados"""

    global dados\_lanzados, ultimo\_dado\_dobles

    with bloqueo\_turnos:

        if not juego\_iniciado:

            enviar\_mensaje(jugador, {"tipo": "error", "mensaje": "El juego aún no ha iniciado."})

            return

        if jugadores[turno\_actual\_idx] != jugador:

            enviar\_mensaje(jugador, {"tipo": "error", "mensaje": "No es tu turno."})

            return

        if dados\_lanzados and not jugador.puede\_relanzar:

            enviar\_mensaje(jugador, {"tipo": "error", "mensaje": "Ya lanzaste los dados en este turno."})

            return

        # Lanzar dados

        dado1 = random.randint(1, 6)

        dado2 = random.randint(1, 6)

        total = dado1 + dado2

        jugador.ultimo\_dado = total

        dados\_lanzados = True

        jugador.puede\_relanzar = False

        # Verificar si son pares

        if dado1 == dado2:

            jugador.pares\_consecutivos += 1

            ultimo\_dado\_dobles = True

            jugador.puede\_relanzar = True

            print(f"[+] {jugador.nombre} sacó pares: {dado1}-{dado2}")

            # Regla de 3 pares consecutivos

            if jugador.pares\_consecutivos >= 3:

                print(f"[!] {jugador.nombre} sacó 3 pares consecutivos")

                enviar\_mensaje(jugador, {

                    "tipo": "info",

                    "mensaje": "¡3 pares consecutivos! Una ficha va a la cárcel. Pasando turno."

                })

                # Enviar ficha a la cárcel

                for i in range(4):

                    if jugador.ficha\_estado[i] in [1, 2]:  # En juego

                        coords\_origen = obtener\_coordenadas\_ficha(jugador, i)

                        jugador.ficha\_estado[i] = 0

                        jugador.ficha\_pos[i] = -1

                        enviar\_a\_todos({

                            "tipo": "movimiento",

                            "color": jugador.color,

                            "ficha\_idx": i,

                            "desde": coords\_origen,

                            "hasta": "carcel"

                        })

                        break

                # Pasar turno

                jugador.pares\_consecutivos = 0

                avanzar\_turno()

                return

        else:

            jugador.pares\_consecutivos = 0

            ultimo\_dado\_dobles = False

        # Obtener movimientos posibles

        movimientos\_posibles = obtener\_posibles\_movimientos(jugador, total)

        # Enviar resultado

        mensaje\_resultado = {

            "tipo": "dados",

            "dado1": dado1,

            "dado2": dado2,

            "total": total,

            "movible\_fichas": movimientos\_posibles,

            "puede\_relanzar": jugador.puede\_relanzar

        }

        enviar\_mensaje(jugador, mensaje\_resultado)

        # Si no hay movimientos posibles

        if not movimientos\_posibles:

            enviar\_mensaje(jugador, {

                "tipo": "info",

                "mensaje": "No tienes movimientos posibles. Pasando turno."

            })

            if not ultimo\_dado\_dobles:

                avanzar\_turno()

def manejar\_mover\_ficha(jugador, ficha\_idx):

    """Maneja el movimiento de una ficha"""

    global dados\_lanzados

    with bloqueo\_turnos:

        if not juego\_iniciado or jugadores[turno\_actual\_idx] != jugador:

            enviar\_mensaje(jugador, {"tipo": "error", "mensaje": "No es tu turno."})

            return

        if not dados\_lanzados:

            enviar\_mensaje(jugador, {"tipo": "error", "mensaje": "Primero lanza el dado."})

            return

        if ficha\_idx < 0 or ficha\_idx >= 4:

            enviar\_mensaje(jugador, {"tipo": "error", "mensaje": "Índice de ficha inválido."})

            return

        # Verificar si el movimiento es válido

        movimientos\_posibles = obtener\_posibles\_movimientos(jugador, jugador.ultimo\_dado)

        if ficha\_idx not in movimientos\_posibles:

            enviar\_mensaje(jugador, {"tipo": "error", "mensaje": "Movimiento no válido."})

            return

        # Ejecutar movimiento

        resultado = ejecutar\_movimiento(jugador, ficha\_idx)

        if resultado:

            # Verificar victoria

            if verificar\_victoria(jugador):

                enviar\_a\_todos({

                    "tipo": "victoria",

                    "ganador": jugador.nombre,

                    "color": jugador.color,

                    "mensaje": f"🎉 ¡{jugador.nombre} ha ganado la partida! 🎉"

                })

                finalizar\_juego()

                return

            # Continuar el juego

            dados\_lanzados = False

            if not jugador.puede\_relanzar:

                avanzar\_turno()

            else:

                enviar\_mensaje(jugador, {

                    "tipo": "info",

                    "mensaje": "Sacaste pares. Puedes lanzar de nuevo o pasar turno.",

                    "puede\_lanzar": True

                })

def ejecutar\_movimiento(jugador, ficha\_idx):

    """Ejecuta el movimiento de una ficha específica"""

    estado = jugador.ficha\_estado[ficha\_idx]

    pos\_actual = jugador.ficha\_pos[ficha\_idx]

    dado\_total = jugador.ultimo\_dado

    coords\_origen = obtener\_coordenadas\_ficha(jugador, ficha\_idx)

    try:

        if estado == 0:  # Sacar de la cárcel

            if not ultimo\_dado\_dobles:

                return False

            jugador.ficha\_estado[ficha\_idx] = 1

            jugador.ficha\_pos[ficha\_idx] = ENTRADAS\_GLOBAL[jugador.color]

            coords\_destino = CAMINO\_GLOBAL[jugador.ficha\_pos[ficha\_idx]]

            enviar\_a\_todos({

                "tipo": "movimiento",

                "color": jugador.color,

                "ficha\_idx": ficha\_idx,

                "desde": "carcel",

                "hasta": coords\_destino

            })

            enviar\_a\_todos({

                "tipo": "info",

                "mensaje": f"{jugador.nombre} sacó una ficha de la cárcel"

            })

        elif estado == 1:  # Mover en camino global

            nueva\_pos = pos\_actual + dado\_total

            # Verificar si entra a la meta

            entrada\_meta = ENTRADAS\_GLOBAL[jugador.color] + len(CAMINO\_GLOBAL) - 7

            if nueva\_pos >= entrada\_meta:

                # Entra al camino de meta

                pasos\_en\_meta = nueva\_pos - entrada\_meta

                if pasos\_en\_meta < len(CAMINOS\_META[jugador.color]):

                    jugador.ficha\_estado[ficha\_idx] = 2

                    jugador.ficha\_pos[ficha\_idx] = pasos\_en\_meta

                    coords\_destino = CAMINOS\_META[jugador.color][pasos\_en\_meta]

                    if pasos\_en\_meta == len(CAMINOS\_META[jugador.color]) - 1:

                        jugador.ficha\_estado[ficha\_idx] = 3

                        jugador.fichas\_en\_meta\_final[ficha\_idx] = True

                        enviar\_a\_todos({

                            "tipo": "info",

                            "mensaje": f"¡{jugador.nombre} llevó una ficha a la meta!"

                        })

                else:

                    return False  # Se pasa de la meta

            else:

                # Continúa en camino global

                if nueva\_pos >= len(CAMINO\_GLOBAL):

                    nueva\_pos = nueva\_pos - len(CAMINO\_GLOBAL)

                jugador.ficha\_pos[ficha\_idx] = nueva\_pos

                coords\_destino = CAMINO\_GLOBAL[nueva\_pos]

                # Verificar captura

                captura = verificar\_captura(jugador, coords\_destino, ficha\_idx)

                if captura:

                    otro\_jugador, otra\_ficha = captura

                    # Enviar ficha capturada a la cárcel

                    otro\_jugador.ficha\_estado[otra\_ficha] = 0

                    otro\_jugador.ficha\_pos[otra\_ficha] = -1

                    enviar\_a\_todos({

                        "tipo": "movimiento",

                        "color": otro\_jugador.color,

                        "ficha\_idx": otra\_ficha,

                        "desde": coords\_destino,

                        "hasta": "carcel"

                    })

                    enviar\_a\_todos({

                        "tipo": "info",

                        "mensaje": f"¡{jugador.nombre} capturó una ficha de {otro\_jugador.nombre}!"

                    })

            # Enviar movimiento

            enviar\_a\_todos({

                "tipo": "movimiento",

                "color": jugador.color,

                "ficha\_idx": ficha\_idx,

                "desde": coords\_origen,

                "hasta": coords\_destino

            })

        elif estado == 2:  # Mover en camino meta

            nueva\_pos\_meta = pos\_actual + dado\_total

            if nueva\_pos\_meta < len(CAMINOS\_META[jugador.color]):

                jugador.ficha\_pos[ficha\_idx] = nueva\_pos\_meta

                coords\_destino = CAMINOS\_META[jugador.color][nueva\_pos\_meta]

                if nueva\_pos\_meta == len(CAMINOS\_META[jugador.color]) - 1:

                    jugador.ficha\_estado[ficha\_idx] = 3

                    jugador.fichas\_en\_meta\_final[ficha\_idx] = True

                    enviar\_a\_todos({

                        "tipo": "info",

                        "mensaje": f"¡{jugador.nombre} llevó una ficha a la meta!"

                    })

                enviar\_a\_todos({

                    "tipo": "movimiento",

                    "color": jugador.color,

                    "ficha\_idx": ficha\_idx,

                    "desde": coords\_origen,

                    "hasta": coords\_destino

                })

            else:

                return False  # Se pasa de la meta

        return True

    except Exception as e:

        print(f"[-] Error ejecutando movimiento: {e}")

        return False

def avanzar\_turno():

    """Avanza al siguiente turno"""

    global turno\_actual\_idx

    turno\_actual\_idx = (turno\_actual\_idx + 1) % len(jugadores)

    enviar\_turno()

def finalizar\_juego():

    """Finaliza el juego actual"""

    global juego\_iniciado

    juego\_iniciado = False

    print("[!] Juego finalizado")

def iniciar\_juego():

    """Inicia una nueva partida"""

    global juego\_iniciado, turno\_actual\_idx, dados\_lanzados

    with bloqueo\_turnos:

        if len(jugadores) >= min\_jugadores\_para\_iniciar and not juego\_iniciado:

            juego\_iniciado = True

            turno\_actual\_idx = 0

            dados\_lanzados = False

            # Resetear estado de jugadores

            for jugador in jugadores:

                jugador.ficha\_estado = [0] \* 4

                jugador.ficha\_pos = [-1] \* 4

                jugador.ultimo\_dado = 0

                jugador.pares\_consecutivos = 0

                jugador.fichas\_en\_meta\_final = [False] \* 4

                jugador.puede\_relanzar = False

            enviar\_a\_todos({

                "tipo": "inicio\_juego",

                "mensaje": "¡El juego ha comenzado!",

                "jugadores": [{"nombre": j.nombre, "color": j.color} for j in jugadores]

            })

            time.sleep(1)

            enviar\_turno()

            print(f"[+] Juego iniciado con {len(jugadores)} jugadores")

            return True

    return False

def manejar\_cliente(conn, addr):

    """Maneja la conexión de un cliente"""

    global juego\_iniciado

    jugador = None

    try:

        # Recibir nombre del jugador

        nombre = conn.recv(1024).decode().strip()

        if not nombre:

            conn.close()

            return

        with bloqueo\_turnos:

            # Verificar disponibilidad

            if len(jugadores) >= max\_jugadores:

                enviar\_mensaje(conn, {"tipo": "error", "mensaje": "Servidor lleno"})

                conn.close()

                return

            if juego\_iniciado:

                enviar\_mensaje(conn, {"tipo": "error", "mensaje": "Juego en curso"})

                conn.close()

                return

            # Crear jugador

            color = colores\_disponibles.pop(0)

            jugador = Jugador(conn, addr, nombre, color)

            jugadores.append(jugador)

            print(f"[+] {nombre} se unió con color {color}")

            # Enviar confirmación

            enviar\_mensaje(jugador, {

                "tipo": "conectado",

                "color": color,

                "mensaje": f"Conectado como {nombre} ({color})"

            })

            # Notificar a otros jugadores

            enviar\_a\_todos({

                "tipo": "jugador\_unido",

                "nombre": nombre,

                "color": color,

                "total\_jugadores": len(jugadores)

            }, except\_jugador=jugador)

            # Iniciar juego si hay suficientes jugadores

            if len(jugadores) >= min\_jugadores\_para\_iniciar:

                iniciar\_juego()

        # Bucle principal del cliente

        while True:

            try:

                mensaje\_raw = conn.recv(1024).decode()

                if not mensaje\_raw:

                    break

                data = json.loads(mensaje\_raw)

                tipo = data.get("tipo")

                print(f"[DEBUG] {jugador.nombre}: {data}")

                if tipo == "lanzar\_dado":

                    manejar\_lanzamiento\_dado(jugador)

                elif tipo == "mover\_ficha":

                    ficha\_idx = data.get("ficha\_idx")

                    manejar\_mover\_ficha(jugador, ficha\_idx)

                elif tipo == "sync\_response":

                    # Manejar respuesta de sincronización

                    pass

                elif tipo == "desconectar":

                    break

            except json.JSONDecodeError:

                print(f"[-] JSON inválido de {jugador.nombre}")

            except Exception as e:

                print(f"[-] Error procesando mensaje de {jugador.nombre}: {e}")

                break

    except Exception as e:

        print(f"[-] Error en conexión de {addr}: {e}")

    finally:

        # Limpiar al desconectar

        if jugador:

            with bloqueo\_turnos:

                if jugador in jugadores:

                    jugadores.remove(jugador)

                    colores\_disponibles.append(jugador.color)

                    colores\_disponibles.sort()

                    enviar\_a\_todos({

                        "tipo": "jugador\_desconectado",

                        "nombre": jugador.nombre,

                        "total\_jugadores": len(jugadores)

                    })

                    print(f"[-] {jugador.nombre} desconectado")

                    # Verificar si se puede continuar el juego

                    if juego\_iniciado and len(jugadores) < min\_jugadores\_para\_iniciar:

                        juego\_iniciado = False

                        enviar\_a\_todos({

                            "tipo": "juego\_pausado",

                            "mensaje": "Juego pausado por falta de jugadores"

                        })

                    elif juego\_iniciado and jugadores:

                        # Ajustar turno si es necesario

                        if turno\_actual\_idx >= len(jugadores):

                            turno\_actual\_idx = 0

                        enviar\_turno()

        try:

            conn.close()

        except:

            pass

def sincronizar\_relojes():

    """Implementa el algoritmo de Berkeley para sincronización"""

    if len(jugadores) < 2:

        return

    print("[🕒] Sincronizando relojes...")

    tiempos = {}

    # Solicitar tiempo a todos los clientes

    for jugador in jugadores[:]:

        try:

            enviar\_mensaje(jugador, {"tipo": "sync\_request"})

            respuesta = jugador.conn.recv(1024).decode()

            data = json.loads(respuesta)

            if data.get("tipo") == "sync\_response":

                tiempos[jugador.nombre] = data.get("tiempo", time.time())

        except Exception as e:

            print(f"[!] Error sincronizando con {jugador.nombre}: {e}")

    if not tiempos:

        return

    # Calcular tiempo promedio

    tiempo\_servidor = time.time()

    tiempos\_lista = list(tiempos.values()) + [tiempo\_servidor]

    tiempo\_promedio = sum(tiempos\_lista) / len(tiempos\_lista)

    # Enviar ajustes

    for jugador in jugadores[:]:

        if jugador.nombre in tiempos:

            ajuste = tiempo\_promedio - tiempos[jugador.nombre]

            try:

                enviar\_mensaje(jugador, {

                    "tipo": "sync\_adjust",

                    "ajuste": ajuste

                })

            except Exception as e:

                print(f"[!] Error enviando ajuste a {jugador.nombre}: {e}")

    print("[✅] Sincronización completada")

def hilo\_sincronizacion():

    """Hilo para sincronización automática"""

    while True:

        time.sleep(30)  # Cada 30 segundos

        if len(jugadores) >= 2:

            sincronizar\_relojes()

def iniciar\_servidor():

    """Inicia el servidor principal"""

    server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

    try:

        server.bind((HOST, PORT))

        server.listen(max\_jugadores)

        print(f"[🎮] Servidor Parqués iniciado en {HOST}:{PORT}")

        # Iniciar hilo de sincronización

        sync\_thread = threading.Thread(target=hilo\_sincronizacion, daemon=True)

        sync\_thread.start()

        while True:

            try:

                conn, addr = server.accept()

                print(f"[+] Nueva conexión desde {addr}")

                cliente\_thread = threading.Thread(

                    target=manejar\_cliente,

                    args=(conn, addr),

                    daemon=True

                )

                cliente\_thread.start()

            except Exception as e:

                print(f"[-] Error aceptando conexión: {e}")

    except KeyboardInterrupt:

        print("\n[!] Servidor detenido por el usuario.")