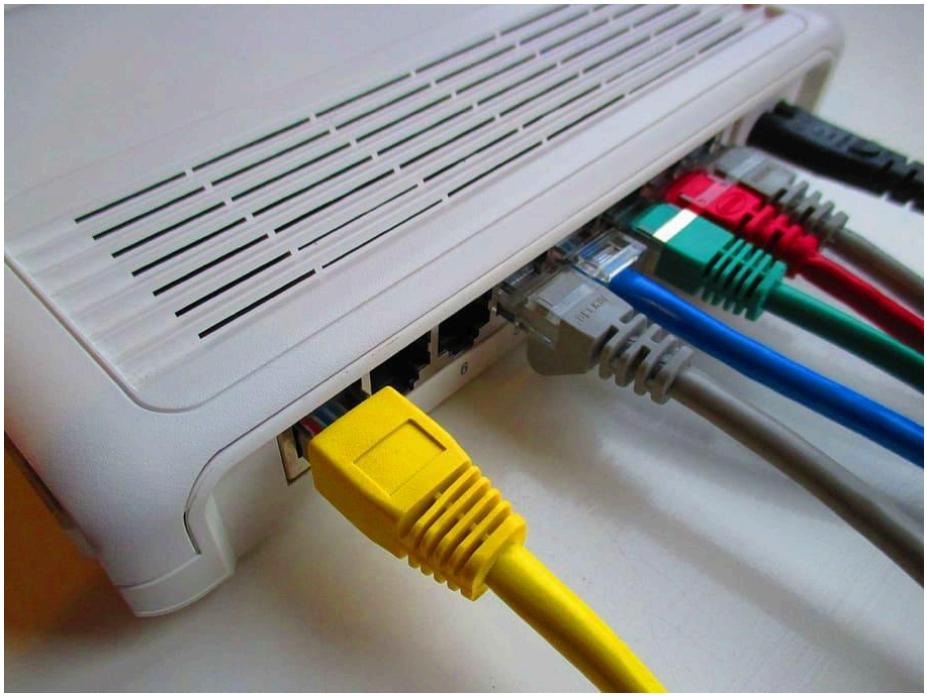
Redes por Computadora

Proyecto cliente/servidor



Profesor: Wilmer Efren Pereira Gonzales

Equipo:
- Leopoldo Morante Castillo
- Patricio Pizaña Vela
- Juan Daniel Rosales Salazar

Misión

La misión de este proyecto es construir un servidor que comunique a varios clientes, los cuales son registrados en una base de datos. Cada cliente tiene la posibilidad de requisitar información sobre los videos ingresados en la base de datos. Estos videos pueden ser descargados por otros clientes mediante una comunicación efectiva y directa con el servidor.
En este proyecto, nos aseguramos de hacer la información lo más clara posible para su visualización, por eso los comandos son claros y concisos. Al igual que los comandos, los mensajes de espera y carga ayudan a que el cliente visualice el proceso de descarga o para que observe la información de contenido en la base de datos. Los comandos son los siguientes:

Para ingresar y descargar videos, el cliente debe registrarse con el servidor insertando el siguiente comando en la consola:

Insc “nombre del cliente” “ip-cliente” “puerto”

En donde “nombre del cliente” es cualquier nombre sin espacios que quiera ingresar y caracterizar a la computadora conectada, el ip-cliente es la ip de esa computadora y el puerto es el puerto de la computadora donde se hará la conexión con la computadora servidor. Una vez registrado el cliente obtendrá una carpeta de videos compartidos y otra de videos recibidos, ahí deberá poner los videos que quiera compartir, al igual que observará los videos descargados.
Una vez registrado, el cliente puede pedir información sobre los videos en la base de datos que fueron ingresados por otros clientes:

Info

También, puede ingresar un video que quiera para poder compartirlo, una vez colocado el video que se desea compartir en la carpeta de videos compartidos después se debe colocar el siguiente comando:

videos “nombre del video”.mp4 o extensión del archivo mandado

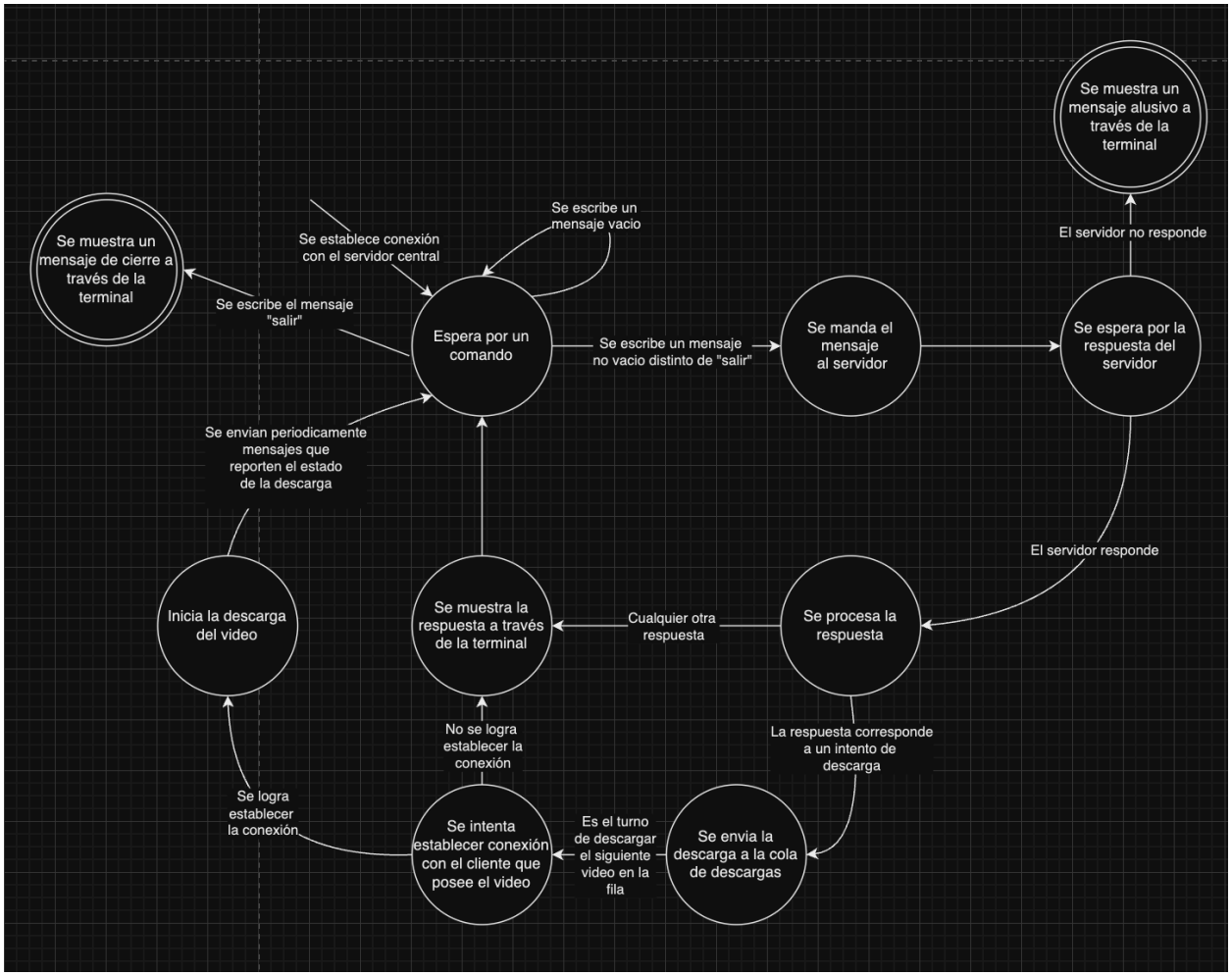
Es necesario ingresar el nombre correcto del video con su extensión para que el video que este en la carpeta sea ingresado de manera correcta a la base de datos.
Finalmente, si se requiere descargar un video por algún cliente se debe ingresar el siguiente comando:

descargar “nombre del video”.mp4 o extensión del archivo mandado

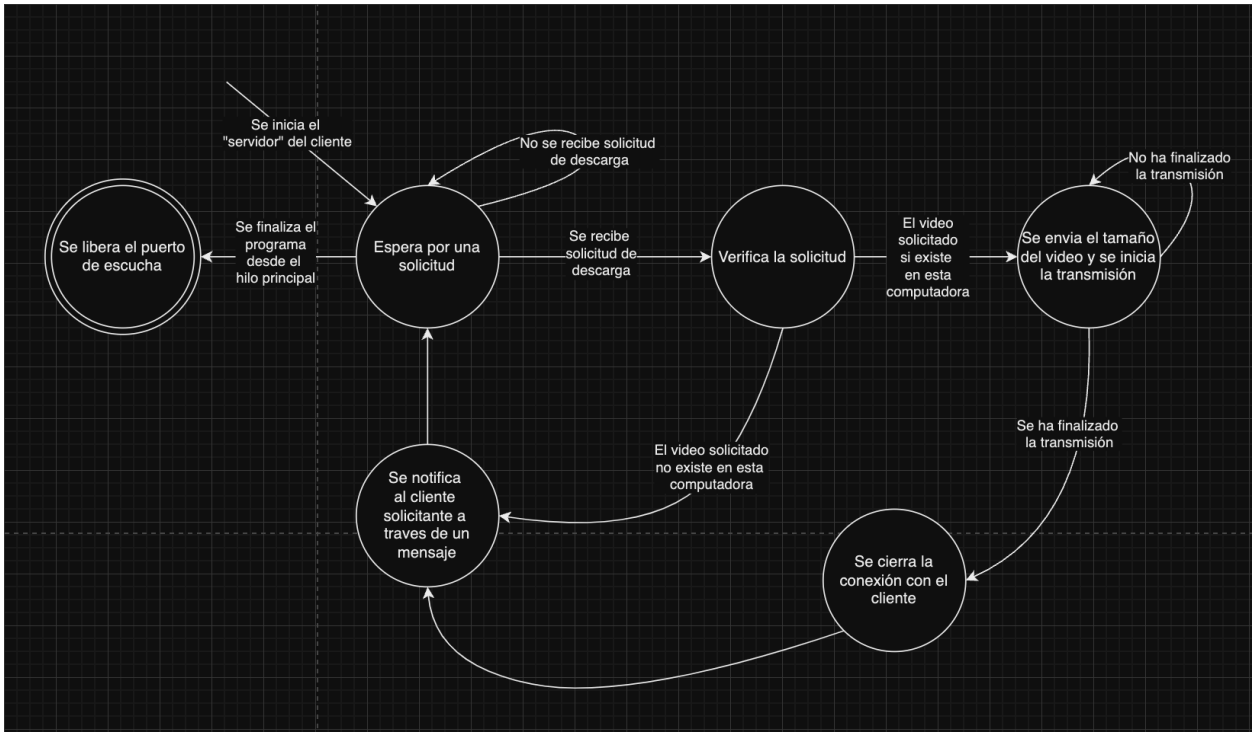
Asegúrese de ingresar el mismo nombre del video que se requiere con la extensión, ya que en otro caso de no ingresar de manera correcta el comando se mandará un mensaje de error advirtiendo al cliente que no se encuentra el video que se requisito.

A continuación se dará a conocer a fondo el código de este proyecto.

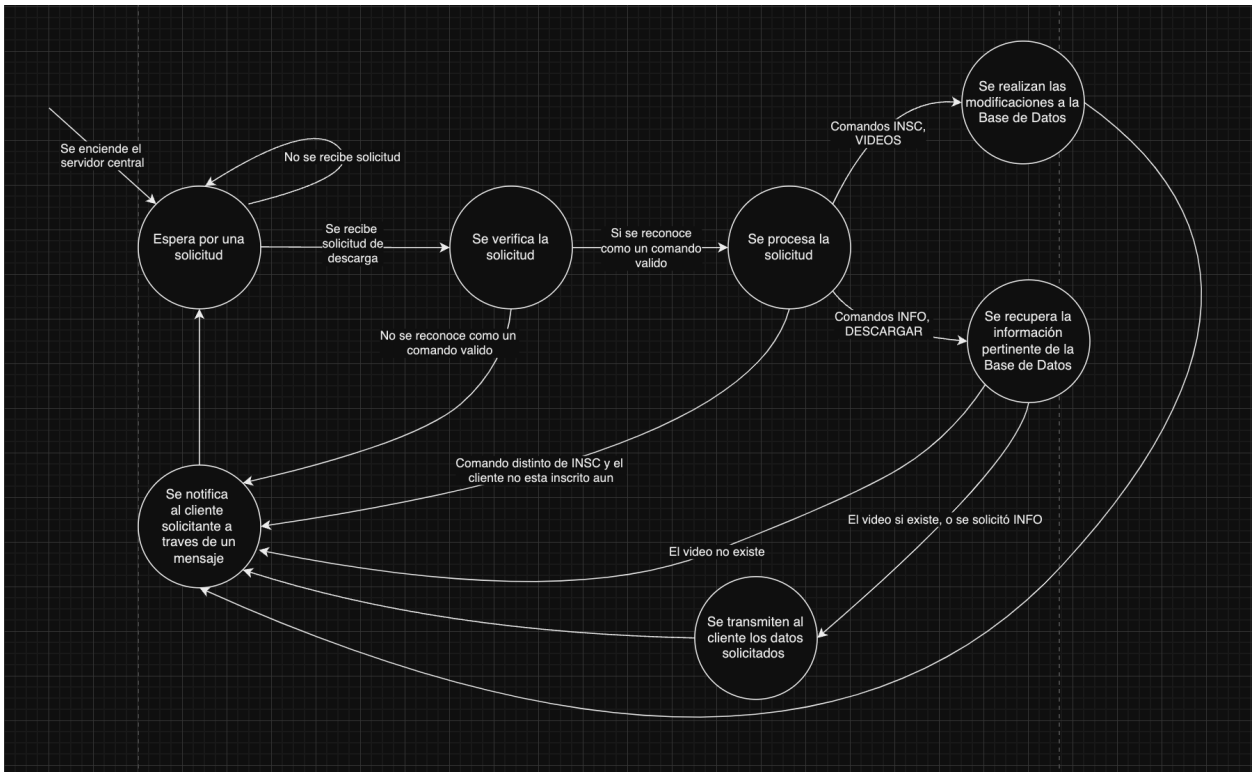
CLIENTE HILO SOLICITANTE DE INFORMACIÓN



CLIENTE HILO PROVEEDOR DE VIDEOS



SERVIDOR CENTRAL



Estructura del código:

Servidor_Central.py

El servidor central cuenta con 3 grandes elementos principales:

1) El ciclo de escucha principal, por dónde el servidor recibe solicitudes de conexión.
2) Las funciones que procesan las solicitudes realizadas por computadoras conectadas.
3) La información de los clientes del servicio, la cual se almacena en un archivo .txt y en una lista de clientes.

1) El servidor central escucha a través de un puerto designado a la espera de que alguna computadora quiera establecer una conexión con él. En cuanto recibe la solicitud, crea un hilo que se encargará de resolver las peticiones del cliente mientras este se mantenga conectado. Se utiliza un socket de tipo TCP y direcciones ipv4.

2) Las solicitudes de cada cliente son atendidas por el hilo correspondiente. La interacción entre servidor y cliente se da a través del envío y recepción de cadenas de caracteres. El análisis de éstas permite tanto al servidor como al cliente procesar adecuadamente las solicitudes del usuario.
Cada que el servidor recibe una solicitud desde un cliente, éste la redirecciona a un método (procesa_solicitud) que verifica si ésta contiene un comando válido (insc, videos, descargar, info) y si el formato de éste es el adecuado. Cada comando tiene una función asociada que resuelve el tipo de comando para el que fue diseñada, pero, de manera general, todas realizan:

a) Una validación del formato del comando. Verifican que el contenido tenga sentido y en caso de no tenerlo, generan una respuesta acorde que posteriormente será enviada al cliente para que pueda reescribir el comando.
b) Una interacción con la base de datos. Dependiendo del comando, se agrega información (insc, videos) o se recupera la ya existente (info,descargar).
c) Un mensaje de respuesta, que será enviado al cliente para notificarle sobre el éxito o falla en el procesamiento de su solicitud.

3) La información de los clientes del servicio se almacena en un archivo llamado ***bd_clientes.txt*** que posee el siguiente formato.
En la primera línea, contiene un número entero no negativo **N** que indica la cantidad de clientes registrados.
Luego, se tiene la información de los N clientes. Para cada uno de ellos se muestra en una primera línea su nombre, ip , puerto de escucha y la cantidad de videos **M** que dice tener.
Las siguientes M líneas constan de una única palabra, el nombre de uno de los videos que posee ese cliente.
Para evitar realizar constantes accesos a la base de datos y facilitar la recuperación y modificación de la información contenida en ella, los datos se leen y almacenan dentro de una lista cada vez que se levanta el servidor. De éste modo, el servidor central, cuando requiere recuperar información accede a la lista, lo cual es más intuitivo y simple de programar. Cada que realiza una modificación sobre la lista, modifica la información guardada en el archivo .txt mediante el uso de una función, de modo que ésta no quede desactualizada.

Cliente.py

El cliente cuenta con 3 elementos principales:

1) El ciclo principal, a través del cual el cliente se comunica con el servidor central.
2) El ciclo de escucha, a través del cual el cliente puede establecer conexiones con otros clientes, las cuales son necesarias para poder llevar a cabo descargas.
3) La cola de descargas y la función que procesa cada una de las descargas.

1) El cliente, conociendo la dirección ip del servidor central, establece una conexión con él. Los detalles de la comunicación son los detallados en el Servidor_Central.py
Al igual que el servidor central, el cliente analiza la cadena que recibe como respuesta a cada una de sus solicitudes. En particular, realiza este análisis para poder determinar cuando se requiere establecer una conexión con otro cliente (cuando se quiere realizar una descarga) y cuando únicamente basta con imprimir en la consola el mensaje recibido desde el servidor.

2) El servidor de escucha cuenta con su propio hilo de ejecución. Se encuentra escuchando a través del puerto 1030 constantemente para recibir solicitudes de conexiones. Ésto es necesario para que el cliente pueda compartir con otros los archivos que posee y que ha publicado en el servidor central. Cuando recibe una solicitud de descarga, verifica la existencia del video dentro de su equipo (en una carpeta que se crea al momento de ejecutar el archivo llamada videos_compartidos), en caso de poseerlo, envía el tamaño del archivo y posteriormente el video solicitado.

3) Los clientes pueden solicitar descargar varios videos de varios dispositivos. Para manejar esto, se posee una cola de descargas, la cual permite el procesamiento secuencial de cada una de las descargas que un cliente desea realizar. Se dedica un hilo a manejar la cola de descargas. Mientras la cola no esté vacía, se toma a la siguiente descarga por realizar, se establece conexión con la computadora (especificada en la descarga) y se realiza la transferencia del archivo. Durante la descarga, se notifica a través de la consola al usuario del status de la descarga.

Documentación del código:

CLIENTE

```
import socket
import threading
import os
import queue
```

```
# Directorios para los videos
DIRECTORIO_VIDEOS = "./videos_compartidos"  # Videos disponibles para compartir
DIRECTORIO_RECIBIDOS = "./videos_recibidos"  # Videos descargados de otros clientes

# Crear los directorios si no existen
os.makedirs(DIRECTORIO_VIDEOS, exist_ok=True)
os.makedirs(DIRECTORIO_RECIBIDOS, exist_ok=True)

# Cola para manejar las descargas
cola_descargas = queue.Queue()

# ------------------------------
# Función para iniciar el servidor local
# ------------------------------
def iniciar_servidor_cliente():
    """
    Inicia un servidor en el cliente para compartir videos con otros clientes.

    El servidor escucha en un puerto predeterminado y responde a solicitudes
    de descarga enviando el contenido de los archivos disponibles en el
    directorio de videos compartidos.

    Args:
        None
    Returns:
        None
    """
    def manejar_solicitud(conn, addr):
        """
        Maneja una solicitud de un cliente para descargar un video.

        Args:
            conn (socket): La conexión activa con el cliente.
            addr (tuple): Dirección del cliente que realiza la solicitud.

        Returns:
            None
        """
        try:
            with conn:
                datos = conn.recv(1024).decode("utf-8")
                if datos.startswith("DESCARGAR"):
                    video = datos.split()[1]
```

```
                ruta_video = os.path.join(DIRECTORIO_VIDEOS, video)
                if os.path.isfile(ruta_video):
                    tamano_archivo = os.path.getsize(ruta_video)
                    conn.sendall(f"{tamano_archivo}\n".encode("utf-8"))

                    # Leer y enviar el contenido del archivo por partes
                    with open(ruta_video, "rb") as archivo:
                        while (chunk := archivo.read(1024)):
                            conn.sendall(chunk)
                else:
                    conn.sendall(b"ERROR El archivo solicitado no existe.\n")
        except Exception as e:
            print(f"Error manejando solicitud: {e}")

    puerto = 1030
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind(("0.0.0.0", puerto))
        puerto = s.getsockname()[1]
        print(f"Servidor del cliente escuchando en el puerto {puerto}")
        s.listen()
        while True:
            conn, addr = s.accept()
            hilo = threading.Thread(target=manejar_solicitud, args=(conn, addr))
            hilo.start()

# ------------------------------
# Función para procesar descargas en cola
# ------------------------------
def procesar_cola_descargas():
    """
    Procesa las descargas en cola de manera secuencial.

    Cada tarea en la cola contiene información del host, puerto y
    nombre del video a descargar.

    Args:
        None
    Returns:
        None
    """
    while True:
        # Obtener la siguiente tarea de la cola
        host, puerto, video = cola_descargas.get()
        try:
```

```
            descargar_video(host, puerto, video)
        except Exception as e:
            print(f"Error en la descarga de '{video}': {e}")
        finally:
            # Marcar la tarea como completada
            cola_descargas.task_done()

# ------------------------------
# Función para descargar un video desde otro cliente
# ------------------------------
def descargar_video(host, puerto, video):
    """
    Descarga un video desde otro cliente.

    Args:
        host (str): Dirección IP del cliente remoto.
        puerto (int): Puerto del cliente remoto.
        video (str): Nombre del archivo a descargar.

    Returns:
        None

    Raises:
        socket.error: Si hay un problema con la conexión al cliente remoto.
        Exception: Para otros errores durante la descarga.
    """
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(10)
            s.connect((host, puerto))
            s.sendall(f"DESCARGAR {video}".encode("utf-8"))

            # Recibir el tamaño del archivo o un mensaje de error
            buffer = b""
            while b"\n" not in buffer:
                buffer += s.recv(1024)
            respuesta = buffer.split(b"\n")[0].decode("utf-8")

            if respuesta.startswith("ERROR"):
                print(f"Error desde el servidor: {respuesta}")
                return

            tamano = int(respuesta)
            print(f"Tamaño del archivo: {tamano} bytes")
```

```
            # Controlar progreso de descarga para evitar redundancias
            primeravez = [True] * 12

            # Descargar el archivo por partes
            with open(os.path.join(DIRECTORIO_RECIBIDOS, video), "wb") as archivo:
                bytes_recibidos = 0
                while bytes_recibidos < tamano:
                    datos = s.recv(1024)
                    if not datos:
                        break
                    archivo.write(datos)
                    bytes_recibidos += len(datos)
                    progreso = (bytes_recibidos / tamano) * 100
                    if int(progreso) % 10 == 0 and primeravez[int(progreso / 10)]:
                        print(f"Descargando {video}: {int(progreso)}% completo")
                        primeravez[int(progreso / 10)] = False

            print(f"Video '{video}' descargado con éxito.")
        except socket.error as e:
            print(f"Error: Se perdió la conexión con el otro cliente: {e}")
        except BrokenPipeError:
            print("Error: La conexión con el servidor se ha cerrado inesperadamente.")
        except Exception as e:
            print(f"Error al descargar el video '{video}': {e}")

# ------------------------------
# Función principal del cliente
# ------------------------------
def cliente():
    """
    Función principal del cliente para gestionar la interacción con otros clientes y el servidor general.

    Este cliente inicia un servidor local para compartir videos, procesa
    descargas en cola y permite al usuario enviar mensajes al servidor general.

    Args:
        None
    Returns:
        None
    """
    # Iniciar el servidor del cliente en un hilo
    hilo_servidor = threading.Thread(target=iniciar_servidor_cliente, daemon=True)
```

```
    hilo_servidor.start()

    # Iniciar el procesador de la cola de descargas en otro hilo
    hilo_cola_descargas = threading.Thread(target=procesar_cola_descargas, daemon=True)
    hilo_cola_descargas.start()

    host = "10.10.17.61"
    puerto = 1025

    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((host, puerto))
            print(f"Conectado al servidor general en {host}:{puerto}")
            while True:
                mensaje = input("Escribe un mensaje (o 'salir' para terminar): ")
                if mensaje.lower() == "salir":
                    print("Cerrando conexión...")
                    break
                if mensaje.strip() == "":
                    continue
                s.sendall(mensaje.encode("utf-8"))
                respuesta = s.recv(1024).decode("utf-8")
                print(f"Respuesta del servidor: {respuesta}")

                if mensaje.upper().startswith("DESCARGAR"):
                    datos = respuesta.split()
                    if len(datos) >= 3 and datos[0] == "EXITO":
                        ip_cliente, puerto_cliente = datos[1], int(datos[2])
                        video = mensaje.split()[1]
                        cola_descargas.put((ip_cliente, puerto_cliente, video))
                        print(f"Descarga de '{video}' añadida a la cola.")
    except socket.error:
        print(f"No se pudo conectar al servidor general en {host}:{puerto}. Verifica si el servidor
está en línea.")
    except KeyboardInterrupt:
        print("\nConexión interrumpida.")
    finally:
        print("Cliente finalizado.")

# Llamar a la función cliente si este script se ejecuta directamente
cliente()
```

SERVIDOR

```
import socket
import threading
from datetime import datetime

# --------------------------------
# Configuración inicial
# --------------------------------
# El servidor está escuchando, por convención, en el puerto 1025
comandos = {"INSC", "VIDEOS", "INFO", "DESCARGAR"}

# Variable global para almacenar clientes registrados
global arr_clientes

# --------------------------------
# Función para registrar logs
# --------------------------------
def printlog(mensaje):
    """
    Registra un mensaje con la hora actual en formato HH:MM:SS.

    Args:
        mensaje (str): Mensaje a registrar.

    Returns:
        None
    """
    hora_actual = datetime.now().strftime("%H:%M:%S")
    print(hora_actual, mensaje)

# --------------------------------
# Clase Cliente
# --------------------------------
class Cliente:
    """
    Representa un cliente registrado en el servidor.

    Attributes:
        nombre (str): Nombre del cliente.
        ip (str): Dirección IP del cliente.
        puerto (int): Puerto en el que escucha el cliente.
        videos (list): Lista de videos asociados al cliente.
    """
    def __init__(self, nombre, ip, puerto, videos):
        self.nombre = nombre
```

```
        self.ip = ip
        self.puerto = puerto
        self.videos = list(videos)

# --------------------------------
# Función para guardar clientes en un archivo
# --------------------------------
def guardar_clientes_en_archivo(clientes):
    """
    Guarda la información de los clientes registrados en un archivo de texto.

    Args:
        clientes (list): Lista de objetos Cliente.

    Returns:
        None
    """
    with open("bd_clientes.txt", "w") as archivo:
        archivo.write(f"{len(clientes)}\n")
        for cliente in clientes:
            archivo.write(f"{cliente.nombre} {cliente.ip} {cliente.puerto} {len(cliente.videos)}\n")
            for video in cliente.videos:
                archivo.write(f"{video}\n")
    printlog("Se modificó la base de datos")

# --------------------------------
# Función para verificar si un cliente está registrado
# --------------------------------
def cliente_registrado(addr, arr_clientes):
    """
    Verifica si un cliente está registrado en la base de datos.

    Args:
        addr (tuple): Dirección IP del cliente (addr[0] contiene la IP).
        arr_clientes (list): Lista de clientes registrados.

    Returns:
        bool: True si el cliente está registrado, False en caso contrario.
    """
    ip = addr[0]
    for cliente in arr_clientes:
        if str(cliente.ip) == str(ip):
            return True
    return False
```

```
# ------------------------------
# Comando: Inscripción
# ------------------------------
def cmd_inscripcion(solicitud, arr_clientes):
    """
    Maneja el comando de inscripción de un nuevo cliente.

    Args:
        solicitud (list): Lista de palabras del comando recibido.
        arr_clientes (list): Lista de clientes registrados.

    Returns:
        str: Respuesta al cliente.
    """
    if len(solicitud) != 4:
        return "Formato de entrada incorrecto\nINSC nom_cliente IP puerto_escucha"

    for c in arr_clientes:
        if c.nombre == solicitud[1]:
            return "Usuario ya registrado"

    c = Cliente(solicitud[1], solicitud[2], solicitud[3], [])
    arr_clientes.append(c)
    guardar_clientes_en_archivo(arr_clientes)
    printlog(f"Se agregó un nuevo cliente: {c.nombre}")
    return f"El cliente {c.nombre} ha sido agregado exitosamente"

# ------------------------------
# Comando: Videos
# ------------------------------
def cmd_videos(solicitud, arr_clientes, addr):
    """
    Maneja el comando para registrar videos asociados a un cliente.

    Args:
        solicitud (list): Lista de palabras del comando recibido.
        arr_clientes (list): Lista de clientes registrados.
        addr (tuple): Dirección del cliente solicitante.

    Returns:
        str: Respuesta al cliente.
    """
    if len(solicitud) < 3:
```

```
        return "Formato de entrada incorrecto\nVIDEOS nom_cliente cantidad_videos lista_videos"

    if cliente_registrado(addr, arr_clientes):
        nombre = solicitud[1]
        nuevos_videos = solicitud[3:]
        for c in arr_clientes:
            if c.nombre == nombre:
                c.videos.extend(nuevos_videos)
                guardar_clientes_en_archivo(arr_clientes)
                printlog(f"Se agregaron videos al usuario {nombre}")
                return "Se agregaron los videos exitosamente"
        return "El cliente proporcionado no existe. Asegúrese de inscribirlo primero"
    return "Regístrese primero"

# ------------------------------
# Comando: Información
# ------------------------------
def cmd_info(solicitud, arr_clientes, addr):
    """
    Proporciona información sobre los videos disponibles en el servidor.

    Args:
        solicitud (list): Lista de palabras del comando recibido.
        arr_clientes (list): Lista de clientes registrados.
        addr (tuple): Dirección del cliente solicitante.

    Returns:
        str: Respuesta al cliente.
    """
    if len(solicitud) != 1:
        return "Formato de entrada incorrecto\nINFO"

    if cliente_registrado(addr, arr_clientes):
        res = "\n".join(video for c in arr_clientes for video in c.videos)
        printlog("Se solicitó información sobre los videos")
        return res if res else "No hay videos registrados"
    return "Regístrese primero"

# ------------------------------
# Comando: Descargar
# ------------------------------
def cmd_descargar(solicitud, arr_clientes, addr):
    """
    Proporciona la información necesaria para descargar un video.
```

```
    Args:
        solicitud (list): Lista de palabras del comando recibido.
        arr_clientes (list): Lista de clientes registrados.
        addr (tuple): Dirección del cliente solicitante.

    Returns:
        str: Respuesta al cliente.
    """
    if len(solicitud) != 2:
        return "Formato de entrada incorrecto\nDESCARGAR nom_video"

    if cliente_registrado(addr, arr_clientes):
        video_buscado = solicitud[1]
        for c in arr_clientes:
            if video_buscado in c.videos:
                printlog(f"Se proporcionó información para la descarga del video {video_buscado}")
                return f"EXITO {c.ip} {c.puerto}"
        return "El video que busca no existe"
    return "Regístrese primero"

# -------------------------------
# Procesa solicitudes de clientes
# -------------------------------
def procesa_solicitud(solicitud, arr_clientes, addr):
    """
    Procesa una solicitud recibida de un cliente.

    Args:
        solicitud (list): Lista de palabras del comando recibido.
        arr_clientes (list): Lista de clientes registrados.
        addr (tuple): Dirección del cliente solicitante.

    Returns:
        str: Respuesta al cliente.
    """
    if solicitud[0].upper() in comandos:
        if solicitud[0].upper() == "INSC":
            return cmd_inscripcion(solicitud, arr_clientes)
        if solicitud[0].upper() == "VIDEOS":
            return cmd_videos(solicitud, arr_clientes, addr)
        if solicitud[0].upper() == "INFO":
            return cmd_info(solicitud, arr_clientes, addr)
        if solicitud[0].upper() == "DESCARGAR":
```

```
            return cmd_descargar(solicitud, arr_clientes, addr)
    return "Comando desconocido"

# ------------------------------
# Maneja comunicación con clientes
# ------------------------------
def manejar_cliente(conn, addr, arr_clientes):
    """
    Maneja la comunicación con un cliente específico.

    Args:
        conn (socket): Objeto socket de la conexión con el cliente.
        addr (tuple): Dirección del cliente (IP y puerto).
        arr_clientes (list): Lista de clientes registrados.

    Returns:
        None
    """
    printlog(f"Conexión establecida desde {addr}")
    with conn:
        while True:
            datos = conn.recv(1024).decode("utf-8")
            if not datos:
                printlog(f"Conexión cerrada por el cliente {addr}")
                break
            respuesta = procesa_solicitud(datos.split(), arr_clientes, addr)
            conn.send(respuesta.encode("utf-8"))

# ------------------------------
# Carga los clientes desde archivo
# ------------------------------
def leer_archivo_y_almacenar():
    """
    Carga los datos de los clientes registrados desde un archivo.

    Returns:
        list: Lista de objetos Cliente.
    """
    clientes = []
    try:
        with open("bd_clientes.txt", "r") as archivo:
            n = int(archivo.readline().strip())
            for _ in range(n):
                linea_cliente = archivo.readline().strip()
```

```
            datos = linea_cliente.split()
            nombre, ip, puerto, mi = datos[0], datos[1], datos[2], int(datos[3])
            videos = [archivo.readline().strip() for _ in range(mi)]
            clientes.append(Cliente(nombre, ip, puerto, videos))
    except FileNotFoundError:
        printlog("El archivo 'bd_clientes.txt' no fue encontrado.")
    return clientes

# ------------------------------
# Función principal del servidor
# ------------------------------
def servidor():
    """
    Función principal para iniciar el servidor.

    Configura el socket para escuchar conexiones de clientes y crea un hilo
    para manejar cada cliente.

    Returns:
        None
    """
    host = "0.0.0.0"
    puerto = 1025

    arr_clientes = leer_archivo_y_almacenar()
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, puerto))
        s.listen()
        printlog(f"Servidor escuchando en {host}:{puerto}")
        while True:
            conn, addr = s.accept()
            hilo = threading.Thread(target=manejar_cliente, args=(conn, addr, arr_clientes))
            hilo.start()

# ------------------------------
# Punto de entrada principal
# ------------------------------
if __name__ == "__main__":
    servidor()
```

Bibliografía:

https://chatgpt.com/share/674406e5-4d14-8003-b7d7-1b14d5dfee68
Notas de clase, Redes de computadoras.