Redes de Computadoras

Curso 2023

Grupo 62

Informe - Obligatorio 2

Autores:
Joaquín Amado
Joaquín Becerra
Mauricio Friss de Kereki

Supervisor:
Ariel Sabiguero Yawelak

October 16, 2023

Contents

A Documentación

La solución a la tarea que será presentada a continuación implementa una arquitectura cliente-servidor con el objetivo de que los clientes puedan consumir un video que les retransmite el servidor que a su vez este consume de una fuente externa. Se utilizan librerías de Python que permiten usar sockets y threads para manejar las conexiones entre los clientes y el servidor.

El programa principal del servidor tiene 3 hilos, uno que se encarga de recibir fragmentos de el video de una fuente externa y ponerlos en una cola que se comparte con todo el programa, el segundo se encarga de retransmitir los fragmentos del video recibido hacia todos los clientes que estén conectados y le hayan indicado al servidor que desean seguir viendo el video, es decir, que no estén pausados. Y por último, un hilo que se creará para cada cliente encargado de interpretar las instrucciones que envíe el cliente. Este último hilo se ejecutará mientras el cliente al que esta asociado permanezca conectado, una vez que se desconecte se terminará.

Por otro lado se tiene el programa del cliente que se encarga de leer los comandos que el usuario escriba en la consola y si son válidos se envían al servidor que es quien los va a interpretar y actuar en consecuencia. El cliente y el servidor se comunican entre si a través de una conexión TCP para enviar las instrucciones de control y a través de una conexión UDP para enviar los fragmentos del video.

El principal problema que tuvimos a la hora de realizar la tarea fue al entender la letra porque nosotros pensamos que el sistema debía ser más complejo ya que debía tener un funcionamiento similar a Netflix, donde el servidor debía recordar cual fue el ultimo fragmento enviado a cada cliente y en caso de pausar poder retomar la retransmisión desde ahí. Cuando fuimos al control intermedio nuestro pseudocódigo modelaba esta realidad lo que nos dificultó bastante a la hora de poder hacer una pequeña demostración del código. Luego de esta instancia cambiamos muchas cosas que ahora si resuelven el problema planteado en la letra de la tarea.
Otro problema que tuvimos fue a la hora de procesar el comando DESCONECTAR porque el programa del cliente se cerraba pero se seguía reproduciendo el

video. Y el problema estaba del lado del servidor a la hora de leer el comando enviado por el cliente.

Finalmente realizamos varias pruebas sobre el código, con varios clientes en simultaneo, ingresando comandos en ordenes incorrectos y utilizando telnet. Como era esperable ocurrieron cosas que no debían pero fueron solucionadas rápidamente y contempladas en las versiones posteriores del código.

B Servidor

```
clientes = {}

def main(server_ip, server_port):
    cola = Queue()
    thread.new(recibirVLC, server_ip, server_port, cola)
    thread.new(enviadorClientes, cola)
    master = socket.tcp()
    master.bind((server_ip, server_port))
    server = master.listen()
    while True:
        client, err = server.accept()
        if(!err)
            thread.new(controladorCliente, client)

    master.close()

def recibirVLC(server_ip, server_port, cola):
    master_vlc = socket.udp()
    master_vlc.bind((server_ip, server_port))
    while True:
        data = master_vlc.receive()
        if data:
            cola.put(data)
        else:
            break
    master_vlc.close()

def controladorCliente(num, client):
    pausado = False
    conectado = False
    buff = ""
    while True:
```

```
data, err = client.receive()
if err:
    client.close()
    break
buff += data

while '\n' in buff:
    #le saco el \n al final de linea
    primer_comando, buff = buff.split('\n', 1)
    print(primer_comando)

    if (primer_comando.startswith("CONECTAR-")):
        # Por si no agrega el puerto al comando
        try:
            puerto = int(primer_comando.replace("CONECTAR-", ""))
        except ValueError:
            ipCliente, puerto = client.getpeer()
        clientes[(ipCliente, puerto)] = True
        conectado = True

        remain = "OK\n"
        while ( remain != "" )
            # Envio respuesta al cliente
            remain , err = client.send("OK\n")

    else:
        match primer_comando:
            case "INTERRUMPIR":
                if conectado and not pausado:
                    pausado = True
                    ipCliente, puerto = client.getpeer()
                    clientes[(ipCliente, puerto)] = False
                    remain = "OK\n"
                    while ( remain != "" )
```

```
                                remain , err = client.send("OK\n")

                        case "CONTINUAR":
                            if conectado and pausado:
                                pausado = False
                                ipCliente, puerto = client.getpeer()
                                clientes[(ipCliente, puerto)] = True
                                remain = "OK\n"
                                while ( remain != "" )
                                    remain , err = client.send("OK\n")
                        case "DESCONECTAR":
                            if conectado:
                                ipCliente, puerto = client.getpeer()
                                clientes.pop((ipCliente, puerto))
                            while ( remain != "" )
                                remain , err = client.send("OK\n")
                            client.close()
                            return
                        case _:
                            continue

def enviadorClientes (num, cola):
    enviador = socket.udp()
    enviador.bind(('127.0.0.1', 65533))
    while True:
        datagrama = cola.get(block = True)
        for cliente in clientes:
            if (clientes[cliente]):
                enviador.sendto(datagrama, cliente)
```

C Cliente

```
def client(server_ip, server_port, vlc_port):
    master = socket.tcp()
    client, err = master.connect((server_ip, server_port))
    if(err)
        master.close()
        return
    paused = False
    connected = False
    while True:
        command = str(input())
        if command == 'CONECTAR':
            command += '-' + str(vlc_port) + '\n'
        else:
            command += '\n'

        if (command.startswith('CONECTAR') != -1 and not connected):
            connected = True
        else:
            match command:
                case 'INTERRUMPIR\n':
                    if(connected and not paused):
                        paused = True
                case 'CONTINUAR\n':
                    if(connected and paused):
                        paused = False
                case 'DESCONECTAR\n':
                    if(connected):
                        connected = False
                case _ :
                    continue

        remain = command
```

```
while ( remain != "" )
    # Envio comando al servidor
    remain , err = client.send(command)

data, err = client.receive()
if "OK" not in data or not connected:
    break
```

References

[1] socket — Low-level networking interface. Recuperado 12/10/2023 https://docs.python.org/3/library/socket.html

[2] threading — Thread-based parallelism. Recuperado 12/10/2023 https://docs.python.org/3/library/threading.html

[3] queue — A synchronized queue class. Recuperado 11/10/2023 https://docs.python.org/3/library/queue.html