Segundo Informe Obligatorio de Redes de Computadoras

Roberto DE ARMAS
Agustin KOLTUKIAN
Matías SARAVIA
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
roberto.de@fing.edu.uy
matias.saravia@fing.edu.uy
agustin.koltukian@fing.edu.uy

I. SOLUCIÓN DEL PROBLEMA A ALTO NIVEL

I-A. Solución para el servidor

Para abordar el problema de crear un servidor que reciba una transmisión de video a través de un puerto UDP y retransmita este vídeo en tiempo real a clientes que se conecten mediante conexiones TCP, se requiere una arquitectura multifuncional con múltiples hilos para gestionar distintos aspectos del servidor.

- **Recepción de Video UDP**: El servidor debe contar con un hilo dedicado para recibir la transmisión de video por UDP desde la fuente VLC. Este hilo escuchará en un puerto UDP específico y gestionará la recepción continua de paquetes de video y futuro envio de datos a los clientes.
- **Gestión de Conexiones TCP**: Otro hilo del servidor se encargará de aceptar las conexiones TCP entrantes de los clientes que deseen recibir la transmisión de video. Este hilo escuchará en un puerto TCP específico y creará un nuevo hilo para atender a cada cliente conectado.
- **Hilos de Clientes**: Cada cliente que se conecta al servidor tiene un hilo dedicado para gestionar su solicitud. Estos hilos se encargarán de:
  - Recibir comandos del cliente: Los hilos de los clientes escucharán los comandos enviados por los clientes, que pueden incluir solicitudes para interrumpir la transmisión de video o desconectar al cliente.
  - Identificación de clientes: Para mantener un registro de los clientes activos, el servidor mantiene una lista que almacena la dirección IP y el puerto de cada cliente. Esta lista sirve para identificar a los clientes y mantener un registro de quiénes están conectados en un momento dado.
  - Desconexión de clientes: Si un cliente decide desconectarse o interrumpir la transmisión, el hilo del cliente se encargará de cerrar la conexión TCP con el cliente y eliminar la entrada correspondiente en la lista de clientes activos.
- **Sincronización de la Lista de Clientes Activos:** Un problema potencial surge cuando varios hilos intentan modificar la lista de clientes activos al mismo tiempo que otro hilo la recorre para enviar datos de video. Para solucionar esto, se implementa un mecanismo de exclusión mutua (mutex) en la lista de clientes. Esto garantiza que solo un hilo pueda modificar la lista a la vez, evitando conflictos de acceso concurrente.
- **Retransmisión de Video UDP a Clientes**: El hilo dedicado a la retransmisión de video UDP recorre la lista de clientes activos y envía paquetes de video a los puertos UDP específicos de cada cliente. Este proceso se realiza de manera concurrente, pero la lista de clientes se protege con un mutex para garantizar que no ocurran problemas de concurrencia. Este hilo es el mismo encargado de la recepción de los datagramas del vídeo.

En resumen, la solución involucra una arquitectura de servidor multiproceso o multihilo con hilos dedicados para la recepción de video UDP, gestión de conexiones TCP y gestión de clientes. La lista de clientes activos se sincroniza mediante un mutex para evitar problemas de concurrencia mientras se retransmite el vídeo a los clientes. Ver anexo donde se deja un pseudocódigo a alto nivel de la solución para el servidor.

I-B. Solucion cliente

La solución para el cliente es más sencilla en comparación con el servidor, ya que su principal tarea es gestionar el intercambio de mensajes con el servidor y controlar la reproducción del video en VLC. Aquí se detalla la funcionalidad y el flujo de operaciones del cliente:

- **Solicitud de Conexión TCP:** El cliente inicia solicitando una conexión TCP con el servidor. Durante esta solicitud, el cliente proporciona al servidor el puerto UDP en el que desea recibir el flujo de video.
- **Conexión TCP Establecida:** Una vez que el cliente ha establecido una conexión TCP con éxito con el servidor, está listo para enviar comandos al servidor y recibir respuestas.

- **Interacción con el Servidor:** El cliente puede enviar los siguientes comandos al servidor:
  - $CONECTAR\,\langle puerto{-}udp{-}cliente\rangle\,\backslash\,n$: Solicita al servidor que inicie el envío del flujo de video al cliente en la dirección IP y puerto UDP especificados.
  - $INTERRUMPIR\,\backslash\,n$: Solicita al servidor que interrumpa temporalmente el envío del flujo de video.
  - $CONTINUAR\backslash n$: Solicita al servidor que reanude el envío del flujo de video. La continuación se realiza a partir del próximo datagrama del flujo recibido por el servidor después de recibir el comando CONTINUAR.
  - $DESCONECTAR\,\backslash\,n$: Solicita al servidor que cancele el envío del flujo de video y, al mismo tiempo, cierra la conexión TCP con el cliente. Esto finaliza la ejecución del cliente.
- **Recepción de Respuestas**: El cliente escucha las respuestas del servidor después de enviar comandos. Las respuestas del servidor pueden incluir confirmaciones de que se han ejecutado los comandos o notificaciones sobre eventos relacionados con la transmisión de video.
- **Control de la Reproducción de VLC**: El cliente también es responsable de controlar la reproducción del video en VLC. Esto puede implicar el inicio de VLC, la apertura del flujo de video recibido en el puerto UDP especificado y la gestión de comandos de reproducción, como reproducir, pausar o detener el video en VLC.
- **Finalización del Cliente**: Cuando el cliente envía el comando $DESCONECTAR\,\backslash\,n$, el servidor detiene la transmisión de video y cierra la conexión TCP con el cliente. En este punto, el cliente puede finalizar su ejecución, lo que podría incluir la terminación de VLC y la liberación de recursos.

La solución para el cliente se centra en la interacción con el servidor, la gestión de los comandos y la reproducción del vídeo en VLC, manteniendo una conexión TCP con el servidor hasta que se solicite la desconexión. Ver anexo donde se deja un pseudocódigo a alto nivel de la solución para el servidor.

I-C. Experimentación

Para verificar el funcionamiento del programa levantamos el servido y al menos un cliente en la misma máquina. Luego, conectamos este cliente al servidor y verificamos la recepción del video. También se comprobó el funcionamiento de los cuatro comandos del protocolo, asegurándose que funcionasen correctamente independientemente del estado previo de la conexión. Luego, se levantaron más clientes y se conectaron al mismo servidor para verificar la capacidad de conexiones en simultáneo soportadas por este. Estas pruebas se hicieron tanto en Windows como en Linux.

I-D. Conclusiones y trabajo futuro

A pesar de ser una implementación bastante sencilla que seguramente tiene mucho margen de mejora, en su estado actual el servidor es capaz de abastecer sin retrasos (suponiendo condiciones de red óptimas) a hasta al menos ocho clientes en simultaneo. Sin embargo, hay varios lugares en donde se pueden realizar mejoras si se quiere mejorar la calidad o rendimiento del servidor. En primer lugar podría buscarse una forma de paralelizar incluso más la carga del servidor con tal de poder soportar más clientes a la vez. Otro tema que podría mejorarse es el manejo de errores, ya que en su actual estado cualquier error que ocurra al manejar un cliente se asume como fallo terminal, cerrando la conexión con el cliente. En su lugar podría tratar de segregar según el tipo de error para ver si es posible recuperase sin cerrar la conexión o no.

II. Anexo

II-A. Pseudocódigo Servidor

Listado 1 Pseudocódigo en Python

```
clientes_activos = []

// Thread transmisor
input = socket.udp()
input.bind('127.0.0.1', 65534)
while true:
   datagrama, ip, port = input.receive()
   transmitir(datagrama)

def transmitir(datagrama):
   output = socket.udp()
   output.bind(server_ip, server_port)
   foreach client in clientes_activos:
      output.sendto(datagrama, client.ip, client.port)
   output.close()

// Thread conexiones
thread.new(cenexiones)
def conexiones():
   master = socket.tcp()
   master.bind(server_ip, server_port)

   server = master.listen()
   while true:
      client, err = server.accept()
      if client != nil:
         thread.new(conexion_cliente, client)

def conexion_cliente(client):
   ip, port = client.getpeer()
   output = {ip}
   state = created
   buffer = []

   while state != closed:
      comando = ''
      while state != closed and !comando:
         data, err = client.receive()
         if err:
            if state == open:
               clientes_activos.remove(output)
            state = closed
         else:
            buffer.add(data)
            indice = buffer.indexof('\n')
            if indice > -1:
               comando = buffer[0:indice]
               buffer.remove_first(indice)

      if comando:
         partes = comando.split('␣')
         switch partes[0]:
            case 'CONECTAR':
               if state != created:
                  break
               output.port = partes[1]
               clientes_activos.add(output)
               state = open
            case 'INTERRUMPIR':
```

```
            if state != open:
               break
            clientes_activos.remove(output)
            state = paused
         case 'CONTINUAR':
            if state != paused:
               break
            clientes_activos.add(output)
            state = open
         case 'DESCONECTAR':
            if state == closed || state == created:
               break
            clientes_activos.remove(output)
            state = closed

      data = 'OK\n'
      while data:
         remain, err = client.send(data)
         data = remain
         if err:
            if state == open:
               clientes_activos.remove(output)
            state = closed
            break

   client.close()
```

II-B. Pseudocódigo Cliente

Listado 2 Pseudocódigo en Python

```
master = socket.tcp()
master.bind(ip_client, port_client)
client, err = master.connect(ip_server, port_server)

if err:
   return

open = true:
while open:
   comando, param = esperar_comando()
   data = param ? comando + '␣' + param + '\n' : comando + '\n'
   while data:
      remain, err = client.send(data)
      data = remain
      if err:
         open = false

   buffer = ''
   while !buffer.contains('OK\n'):
      data, err = client.receive()
      buffer.add(data)
      if err:
         open = false

client.close()
master.close()
```

REFERENCIAS

[1] API de sockets para un lenguaje de alto nivel. *Cartilla de Sockets*. Disponible en: https://eva.fing.edu.uy/pluginfile.php/254645/mod_resource/content/0/cartilla_sockets.pdf

[2] VideoLAN Organization. *Página oficial de VLC*. Disponible en: https://www.videolan.org/vlc/