Laboratorio 3: Actividad 2 – Implementación de cliente y servidor UDP

Grupo 4 - Lab Sección 1:

Martín Ubaque
Kevin Babativa
Daniel Reyes

Links relevantes:

- **Repositorio GitHub con aplicaciones Servidor y Cliente:** https://github.com/MartinUbaque/LabUDP
- **Enlace a vídeo sobre desarrollo y funcionamiento:** **VideoUDP.mp4**
- **Enlace de descarga de las capturas de tráfico:** **Capturas Wireshark**

4.2 Servidor de Transferencia de Archivos

Para el servidor usamos, entre otras, 2 librerías principales: Socketserver para que el servidor sea capaz de manejar múltiples llamadas de distintos usuarios, y OS cuya función es poder acceder a servicios del sistema operativo. También, se define en primera instancia cuál va a ser la dirección IP del servidor y el puerto por el cuál se recibirán las peticiones que será el 8888.

```
import socketserver
import os
from datetime import datetime
import time

ServerAddress = ("127.0.0.1", 8888)
```

Ahora bien, creamos una clase llamado *MyUDPRequestHandler* cuyo principal propósito es manejar todo lo concerniente al protocolo UDP. En primer lugar, vamos a querer saber cuál archivo es el que vamos a enviar. Posteriormente, tenemos 2 métodos los cuáles son:

- **Log:** El propósito de este método es definir la manera en que se generará el archivo con el log correspondiente.

```
class MyUDPRequestHandler(socketserver.DatagramRequestHandler):
    archivo = input('Ingrese el archivo: ')
    def log(self, archivo, cliente, tiempo, confirmacion):
        nombreLog =
atetime.today().strftime('%Y-%m-%d-%H-%M-%S')+'-log.txt'
        size = os.path.getsize(archivo)
        with open('Logs/'+nombreLog, 'a+') as f:
            f.write(f'Archivo: {archivo} size: {size} cliente {cliente}
tiempo: {str(tiempo)} ms tuvo resultado {confirmacion}.\n')
```

- **Handle:** Este método sirve para generar el socket correspondiente a la solicitud, definir el datagrama, el tamaño del buffer y cargar el archivo a enviar al programa. Posteriormente se envía la data dentro de un bucle que enviará los datagramas mediante el socket mientras aún falten. Finalmente se genera la confirmación y se imprime en el log.

```
def handle(self):
        socket = self.request[1]
        archivo = self.archivo
        datagram = self.rfile.readline().strip()
        file = open(archivo, 'rb')
        buf=1024
        start = time.time()
        data = file.read(buf)
        while(data):
            socket.sendto(data, self.client_address)
            data = file.read(buf)
        end = time.time()
        confirmacion = 'OK'
        self.log(archivo, self.client_address, end-start, confirmacion)
        print(f'Se envió el archivo al cliente {self.client_address}')
```

Para finalizar el programa encargado del servidor, se crea el objeto *UDPServerObject* mediante llamar a la librería *Socketserver* que habíamos importado previamente, usando su método *ThreadingUDPServer* y pasando como parámetro la IP del servidor y la clase *MyUDPRequestHandler* que habíamos mencionado previamente. Finalmente, llamamos al método *server_forever* de ese objeto que mantendrá encendido y funcionando el servidor a la espera de cualquier solicitud de algún cliente.

```
UDPServerObject = socketserver.ThreadingUDPServer(ServerAddress,
MyUDPRequestHandler)
UDPServerObject.serve_forever()
```

4.3 Cliente UDP

Para el cliente usamos las librerías socket, threading y os para manejar las conexiones mediante sockets, crear los hilos para los distintos clientes concurrentes y acceder a los servicios del sistema operativo, respectivamente.

Ahora bien, para iniciar nosotros declaramos las variables que representarán el saludo al servidor, el tamaño del buffer y la dirección IP del servidor al que nos vamos a conectar junto a su puerto que era el 8888.

```
msgFromClient       = "Hello UDP Server"
bytesToSend         = str.encode(msgFromClient)
bufferSize          = 1024
serverAddressPort   = ("127.0.0.1", 8888)
```

Para manejar la concurrencia de los clientes tenemos el siguiente fragmento de código que se encarga de preguntarle al usuario cuantos clientes desea, crearlos y guardarlos en una lista para luego iterar sobre ellos y efectuar su conexión al servidor.

```
ThreadList  = []
ThreadCount = int(input('Ingrese el número de clientes: '))

for index in range(ThreadCount):
    ThreadInstance = threading.Thread(target=Connect2Server,
args=(index, ThreadCount))
    ThreadList.append(ThreadInstance)
    ThreadInstance.start()

for index in range(ThreadCount):
    ThreadList[index].join()
```

En el programa poseemos dos métodos principales:

- **Log:** Realiza el mismo papel que el log del programa del servidor y funciona de la misma manera.

```
def log(archivo, cliente, tiempo, confirmacion):
        nombreLog =
datetime.today().strftime('%Y-%m-%d-%H-%M-%S')+'-log.txt'
        size = os.path.getsize(archivo)
        with open(nombreLog, 'a+') as f:
            f.write(f'Archivo: {archivo} size: {size} cliente {cliente}
tiempo: {str(tiempo)} ms tuvo resultado {confirmacion}.\n')
```

- **Connect2Server:** Este método se encarga de generar el archivo una vez lo recibe, también de estipular el tiempo de Timeout, y finalmente de generar el socket y procesar el recibimiento de los datagramas enviados por el socket desde el servidor. Esto lo hace mediante un ciclo el cuál esperará la confirmación del servidor para finalizar el proceso.

```
def Connect2Server(nClient, nConexiones):
    nombreArchivo = f'Cliente{nClient}-Prueba-{nConexiones}.txt'
    timeout = 3
    UDPClientSocket = socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM)
    UDPClientSocket.sendto(bytesToSend, serverAddressPort)
    start = time.time()
    f = open(nombreArchivo, 'wb+')
    while True:
        ready = select.select([UDPClientSocket], [], [], timeout)
        if ready[0]:
            data, addr = UDPClientSocket.recvfrom(bufferSize)
            f.write(data)
            confirmacion = 'NO OK'
        else:
            f.close()
            confirmacion = 'OK'
            break
    end = time.time()
    log(nombreArchivo, nClient, end-start, confirmacion)
```

4.4 Análisis de desempeño del servicio

A continuación, podemos encontrar la tabla que resume las pruebas del análisis de desempeño que se realizó. No obstante, hay 3 columnas que se omitieron ya que sus valores se repiten constantemente en todas las filas, las cuáles son los siguientes:

- Transferencia exitosa: Sí. En todas las pruebas lograron enviarse una cantidad importante de bytes.
- Valor total de bytes transmitidos por el servidor a cada cliente: La cantidad total de bytes que transmitió el servidor fue igual al tamaño total del archivo. Por lo cuál, esa información ya la podemos encontrar en la columna "*Tamaño del archivo (Bytes)*".
- Número Puerto utilizado para la conexión de cada cliente: Desde la programación para el cliente se estipuló que siempre se usaría el puerto 8888, por lo que ese valor no variará en ninguna prueba.

	Tamaño del archivo (Bytes)	Número de conex.	Cliente	Valor total de bytes recibidos por cada cliente	Tiempo de transf. en seg	Tasa de transf. a cada cliente (MB/s)	Puerto conexión con cada cliente (Aplicación Servidor)	% de datos que recibidos por el cliente
Prueba 1	104857600	1	1	104712192	4,076	25,689	64927	99,86%
Prueba 2	262144000	1	1	262139904	7,013	37,378	52993	100,00%
Prueba 3	104857600	5	1	51982336	7,444	6,983	58754	49,57%
			2	53821440	7,459	7,216	58755	51,33%
			3	51902464	7,460	6,958	58756	49,50%
			4	53906482	7,988	6,749	58757	51,41%
			5	50154102	7,124	7,040	58758	47,83%
Prueba 4	262144000	5	1	132996096	12,552	10,595	56378	50,73%
			2	120783872	12,553	9,622	56379	46,08%
			3	115772416	12,553	9,222	56380	44,16%
			4	124101632	12,583	9,863	56381	47,34%
			5	120783872	12,553	9,622	56382	46,08%
Prueba 5	104857600	10	1	53516288	11,445	4,676	60178	51,04%
			2	51369984	11,459	4,483	60179	48,99%
			3	50455552	11,458	4,403	60180	48,12%
			4	53552128	11,460	4,673	60181	51,07%
			5	53285888	11,458	4,650	60182	50,82%
			6	48602112	11,477	4,235	60183	46,35%

			7	48768000	11,477	4,249	60184	46,51%
			8	49394002	11,458	4,311	60185	47,11%
			9	51938110	11,460	4,532	60186	49,53%
			10	52934120	11,477	4,612	60187	50,48%
Prueba 6	262144000	10	1	123635712	19,874	6,221	64918	47,16%
			2	134190080	19,888	6,747	64919	51,19%
			3	135300096	19,904	6,798	64920	51,61%
			4	125400064	19,903	6,301	64921	47,84%
			5	133257216	19,905	6,695	64922	50,83%
			6	122386432	19,917	6,145	64923	46,69%
			7	127987712	19,935	6,420	64924	48,82%
			8	125410304	19,931	6,292	64925	47,84%
			9	120757248	19,935	6,057	64926	46,07%
			10	125030400	19,932	6,273	64927	47,70%

Analizando las pruebas realizadas podemos notar los siguientes aspectos más relevantes de los datos:

- **Porcentaje de datos recibidos por el cliente:** Podemos ver que en la ***Prueba 1*** y ***Prueba 2*** el porcentaje de datos que fueron enviados y recibidos exitosamente por el cliente son los más altos de todas las pruebas. En el resto de pruebas, los clientes recibieron alrededor del 50% de los datos que les fueron enviados. Esto pudo deberse a que los clientes compartían el puerto 8888 como mencionamos previamente, por lo que al tener que compartir el puerto al mismo tiempo con y el envío de un archivo igual, muchos de estos paquetes enviados se perdieron reduciendo la cantidad total de bytes recibidos por los clientes.

- **Tasa de transferencia a cada cliente (MB/s):** De igual forma, podemos notar en la ***Prueba 1*** y ***Prueba 2*** la tasa de transferencia a cada cliente fueron las más alta de las pruebas. Y por otro lado, las tasas de transferencia más bajas las tuvo la ***Prueba 5***, donde se conectaron 10 clientes al servidor. Esto demuestra lo que comentamos en el literal anterior, que el uso compartido del mismo puerto al tiempo pudo haber ocasionado un retardo en la red que también pudo haber generado la pérdida de paquetes mencionada.

- **Tiempo de transferencia en segundos:** Podemos evidenciar cómo el tiempo de transferencia en segundos del archivo aumenta en relación a la cantidad de clientes conectados y al tamaño del archivo.

Resultados de TCP vs UDP: Los factores más importantes a tener en cuenta para comparar ambos resultados son los siguientes:

- **Tiempo de transferencia del archivo:** Podemos notar que el tiempo de transferencia del archivo en UDP es menor al que recibimos en TCP. Esto se debe a que el protocolo de UDP es más simple que el de TCP, sin necesitar realizar algún tipo de validación sobre el recibimiento de los datos.
- **Porcentaje de datos recibidos por el cliente:** En TCP es fundamental que no exista pérdida de paquetes, para eso el protocolo cuenta con las validaciones que mencionamos previamente. Es por eso que durante la ejecución de estas pruebas pudimos notar que la pérdida de paquetes estuvo presente, y dependiendo del escenario se aumenta o disminuye el porcentaje de pérdida de paquetes.

5. Preguntas

1) **Si tuviera que desarrollar un servicio de streaming de video con una arquitectura Cliente/Servidor donde la transmisión fuera en multidifusión. Mencione cuales serían las consideraciones técnicas que tendría en cuenta para el desarrollo del servicio, sea lo más detallado posible**

   Es necesario un protocolo de enrutamiento múltiple, es necesario que el servidor le envíe la información a un enrutador junto a un enrutador multicast le envía la misma información a todo el grupo involucrado. Así mismo, el emisor debe conocer los destinatarios de forma previa, de esta forma es como se forman los grupos de multicasting.

2) **¿Es posible desarrollar aplicaciones UDP que garanticen la entrega confiable de archivos? Qué consideraciones deben tenerse en cuenta para garantizar un servicio de entrega confiable utilizando dicho protocolo. Justifique su respuesta.**

   UDP Por sí mismo provee un checksum para comprobar la integridad de los datagramas, sin embargo, esto no garantiza que todos los datagramas lleguen al cliente, por lo cual sería necesario marcar los paquetes y en una primera interacción con el cliente informarle cuántos datagramas se van a enviar, en cada datagrama va la información de cual datagrama es, por lo cual al final de la transmisión se le puede pedir al servidor que re-envía los datagramas faltantes.

3) **¿Se podrían considerar servicios de emisión de contenidos que funcionen con el protocolo TCP? Justifique su respuesta.**

   Si, sería posible tener servicios de emisión sobre TCP, sin embargo, estos servicios no serían ideales ya que se tendría que esperar todo el buffering de la recepción de los archivos, esto haría que la calidad de la información fuese la máxima, sin embargo, este proceso es más lento que en UDP (En TCP Se espera el tiempo de reenvío de paquetes).