Laboratorio 3: comunicación de mensajes por TCP y UDP

Juan Bello Durango

Arley Bernal Muñeton

Kevin Garay Gaitan

Sistemas Distribuidos

Profesor: John Corredor Franco

1. Introducción

En el contexto de los sistemas distribuidos, la comunicación entre procesos a través de la red es un pilar fundamental. Dos de los protocolos más utilizados para este propósito son TCP (Transmission Control Protocol) y UDP (User Datagram Protocol). Aunque ambos permiten la transferencia de datos entre cliente y servidor, sus características son muy diferentes: TCP garantiza la fiabilidad y el orden de los mensajes, mientras que UDP ofrece mayor velocidad y simplicidad, pero sacrifican la confiabilidad.

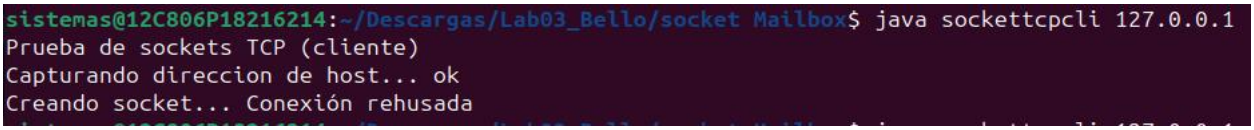
Este taller busca comparar de manera práctica el comportamiento de ambos protocolos, evidenciando sus ventajas, limitaciones y aplicaciones más adecuadas.

2. Batería de pruebas

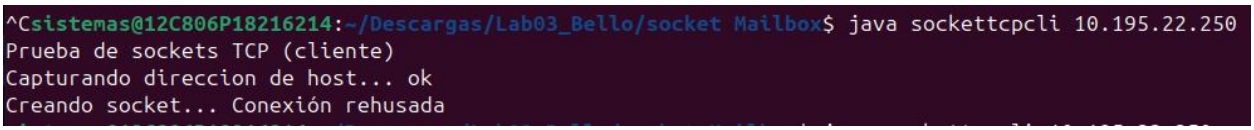
Caso 1: Cliente TCP sin servidor

El cliente intenta establecer conexión, pero no encuentra un servidor escuchando en el puerto especificado.

Local:



Remoto:



Resultado: Se genera una excepción indicando que la conexión fue rechazada y el programa termina.

Esto se debe a que TCP requiere establecer una conexión antes de transmitir datos. Si no hay un servidor disponible, el cliente no puede avanzar y el programa notifica el error. Esto debido a la naturaleza confiable y orientada a conexión de TCP.

Caso 2: Cliente TCP con servidor (caso ideal)

El servidor TCP se ejecuta primero y luego el cliente se conecta y envía un mensaje.

Cliente local:

```
sistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$ java sockettcpcli 127.0.0.1
Prueba de sockets TCP (cliente)
Capturando direccion de host... ok
Creando socket... ok
Introduce mensajes a enviar:
hola
prueba 2
correcto
```

Server local:

```
sistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$ java sockettcpser
Prueba de sockets TCP (servidor)
hola
prueba 2
correcto
null
```

Cliente remoto:

```
sistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$ java sockettcpcli 10.195.22.250
Prueba de sockets TCP (cliente)
Capturando direccion de host... ok
Creando socket... ok
Introduce mensajes a enviar:
hola
conexion remota
correcto
```

Server remoto:

```
sistemas@12C806P11216216:~/Descargas/socket Mailbox$ javac *
sistemas@12C806P11216216:~/Descargas/socket Mailbox$ java sockettcpser
Prueba de sockets TCP (servidor)
hola
conexion remota
correcto
null
sistemas@12C806P11216216:~/Descargas/socket Mailbox$
```

Resultado: La conexión se establece exitosamente, el servidor recibe el mensaje correctamente. A demás observamos que en ambos casos al terminar la ejecución del cliente, el servidor detecta la salida y también termina

TCP garantiza la entrega ordenada y confiable de los datos. Aquí se observamos la correcta sincronización cliente-servidor, con un flujo de comunicación seguro. Este es

el escenario ideal y representa el uso típico de TCP en aplicaciones como HTTP, FTP o correo electrónico.

Caso 3: Cliente UDP sin servidor

El cliente envía un datagrama a la dirección y puerto de un servidor que no está en ejecución.

Local:

```
sistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$ java socketudpcli 127.0.0.1
Prueba de sockets UDP (cliente)
Creando socket... ok
Capturando direccion de host... ok
Introduce mensajes a enviar:
hola
```

Remoto:

```
^Csistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$ java socketudpcli 10.195.22.250
Prueba de sockets UDP (cliente)
Creando socket... ok
Capturando direccion de host... ok
Introduce mensajes a enviar:
mensaje sin server
1
2
mensaje con server
5
8
hola
```

Resultado: El mensaje simplemente se envía, pero no recibe respuesta alguna, no hay error inmediato pero los mensajes se pierden.

A diferencia de TCP, UDP no establece conexión ni verifica si el servidor está disponible. El datagrama se lanza a la red, aunque nadie lo reciba. Esto debido a que UDP es no orientado a conexión y no garantiza la entrega de los datos.

Caso 4: Cliente UDP con servidor

El servidor UDP se ejecuta primero, luego el cliente se conecta y envía un mensaje que llega correctamente.

En el caso del servidor remoro ejecutamos primero el cliente UDP, mandamos algunos mensajes, y luego ejecutamos el servidor para comprobar la perdida de mensajes.

Cliente local:

```
^Csistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$ java socketudpcli 127.0.0.1
Prueba de sockets UDP (cliente)
Creando socket... ok
Capturando direccion de host... ok
Introduce mensajes a enviar:
test
llegado
sigue mandando sin servidor
```

Server local:

```
sistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$ java socketudpser
Prueba de sockets UDP (servidor)
Creando socket... ok
Recibiendo mensajes...
test
llegado
^Csistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$
```

Cliente remoto:

```
^Csistemas@12C806P18216214:~/Descargas/Lab03_Bello/socket Mailbox$ java socketudpcli 10.195.22.250
Prueba de sockets UDP (cliente)
Creando socket... ok
Capturando direccion de host... ok
Introduce mensajes a enviar:
mensaje sin server
1
2
mensaje con server
5
8
hola
```

Server remoto:

```
sistemas@12C806P11216216:~/Descargas/socket Mailbox$ java socketudpser
Prueba de sockets UDP (servidor)
Creando socket... ok
Recibiendo mensajes...
mensaje con server
5
8
hola
```

Resultado: Se establece una comunicación rápida y eficiente, aunque sin verificación estricta de orden o confiabilidad pues los mensajes remotos enviados antes de la ejecución del servidor se pierden por completo.

La comunicación ocurre sin necesidad de handshake previo. haciéndolo adecuado para aplicaciones donde la velocidad es más importante que la confiabilidad, como streaming, VoIP o videojuegos en línea.

3. Conclusión

El taller permitió evidenciar de manera práctica las diferencias esenciales entre TCP y UDP, pues TCP ofrece confiabilidad, control de errores y asegura el orden de los datos, pero introduce mayor complejidad y latencia. Mientras que UDP es más rápido y simple, pero no garantiza la entrega ni el orden de los mensajes.

Los experimentos con cliente sin servidor mostraron cómo TCP notifica activamente los errores de conexión, mientras que UDP simplemente ignora la falta de receptor. En los casos ideales con servidor presente, TCP brindó una comunicación confiable y estructurada, mientras que UDP facilitó un intercambio rápido y ligero.