Informe TP 4

Estudiante: Bautista Frigolé

Fecha: 11/04/2024

Patrón Request-Reply

5. a. **Cambiar el ejemplo para que el mensaje sea diferente / la respuesta del servidor sea diferente.**

Server

```
const zmq = require("zeromq")

async function run() {
 const sock = new zmq.Reply

 await sock.bind("tcp://*:3000")
 console.log('Reply server iniciado')

 for await (const [msg] of sock) {
   console.log(`Recibido: ${msg.toString()}`)
   const reply = ""

   if (msg.toString() === "Hola") {
     reply = "Hola, cómo estás?"
   } else {
     reply = "Chauuu"
   }

   console.log("Enviando: ", reply)
   await sock.send(reply)
   console.log(reply)
 }
}

run()
```

Client

```
const zmq = require("zeromq")
const config = require("../config")

async function run() {
 const sock = new zmq.Request

 sock.connect(`tcp://${config.server_host}:3000`)
 console.log(`Cliente conectado (?) al server
${config.server_host}`)
  await sock.send("Hola")
```

```
 var [result] = await sock.receive()
 console.log(result.toString())

 await sock.send("Bieeen, me tengo que ir después la seguimos")
 var [result] = await sock.receive()
 console.log(result.toString())
}

run()
```

b. **Detener el servidor y ejecutar el cliente, ¿que pasó? Luego ejecutar el servidor. ¿Qué pasó con el cliente? ¿Esto es lo mismo en otros protocolos conocidos? ¿Por qué cree que pasa esto?**

Al detener el servidor y ejecutar el cliente, el cliente se conecta y envía el mensaje pero nunca recibe una respuesta del servidor, se queda tildado esperando respuesta. Si luego ejecutamos el servidor, inmediatamente éste responde al cliente y continúa la ejecución normal. En otros protocolos como HTTP no sucedía así debido a que TCP tiene un tipo de comunicación persistente y no transitoria.

c. **Investigue cómo listar los puertos abiertos tanto en la máquina que hace de cliente como la que hace de servidor con solo un programa andando. ¿Cuál es la diferencia entre ambos?**

Se listaron con el comando *ss -tulpn.* La diferencia es que la máquina servidor está escuchando en el puerto 3000 y la máquina cliente no.

Patrón Publisher-Subscriber

1. **¿Qué pasa si solo apago uno de los dos programas?**
   Si se apaga el publisher, el subscriber se sigue ejecutando pero deja de recibir mensajes. Si se apaga el subscriber, el publisher sigue enviando sus mensajes sin cambiar nada.

   a. **¿Si no hay cliente, qué pasa con los mensajes publicados?**
      Nadie los recibe por lo que esos mensajes se pierden y no son guardados en ningún lado.

   b. **¿Hay algo que te llame la atención?**
      Me llama la atención que los subscriber no reciben el primer mensaje del publisher.

2. **Probá ejecutando dos o más subscriber en dos consolas distintas.**
   Ambos reciben los mensajes del publisher.

3. **Ahora intente probar ejecutar dos publisher en el servidor. ¿Funciona? ¿Por qué no?**
   Nos lanza un error ya que ya se está utilizando la dirección 'tcp://*:3000'.

4. **¿Qué pasa si cambio el tópico en el cliente o en servidor?**
   Si se cambia el tópico, el publisher y el subscriber ya no se pueden comunicar.

5. **¿Puede un pub tener más de un tópico en ejecución? Y un subscriber atender múltiples tópicos?**
   Sí, puede un publisher tener más de un tópico en ejecución, y también puede un subscriber atender múltiples tópicos.

Patrón Push-Pull

1. **¿Qué pasa si solo apago uno de los dos programas? ¿Qué diferencia esto del patrón pub-sub?**
   Si se apaga alguno de los dos programas, detienen la ejecución hasta poder restablecer la comunicación. La diferencia con el publisher, es que el producer no envía mensajes hasta que haya un worker escuchando del otro lado.

2. **Probá ejecutando dos o más worker en dos consolas distintas. De nuevo, ¿que diferencia esto del patrón pub-sub?**
   La diferencia con el patrón pub-sub es que los subscribers recibían todos los mensajes que el publisher enviaba, en este caso, el producer reparte los mensajes entre los workers, envía “intercaladamente” a cada uno.

Probando portabilidad

Se implementó un subscriber en Python:

```
import zmq

context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect("tcp://localhost:5555")

socket.setsockopt_string(zmq.SUBSCRIBE, "")

while True:
   message = socket.recv_string()
   print("Mensaje recibido:", message)
```