TSR - Práctica 2: **0MQ**

Curso 2019/20

Contents

1. Introducción

1.1. Objetivos

- Afianzar los conceptos teóricos introducidos en el tema 3
- Experimentar con distintos patrones de diseño (patrones básicos de comunicación) y tipos de sockets
- Profundizar en el patrón broker (proxy inverso)

1.2. Propuesta para la organización del tiempo

- Sesión 1.- Publicador rotatorio, prueba aplicación Chat
- Sesión 2.- Parametrización broker, estadísticas broker
- Sesión 3.- Broker para clientes + broker para workers
- Sesión 4.- Estudio posibles ampliaciones broker, Prueba patrón broker tolerante a fallos

1.3. Método de trabajo

- Para simplificar las pruebas, lanzamos los distintos componentes de una aplicación en la misma máquina (utilizando 'localhost' como IP del servidor). No obstante, todo el código suministrado podría probarse sobre máquinas distintas interconectadas sin apenas cambios
- Los números de port que aparecen en los ejemplos se deben modificar según las instrucciones indicadas en el boletín de la primera práctica
- Todos los fragmentos de código proporcionados deberían probarse en el laboratorio. Por brevedad, los fragmentos de código incluyen valores fijos para indicar port, host, mensaje a escribir, etc: en código real deberían leerse esos valores desde línea de órdenes

- El fichero `RefZMQ.pdf` contiene material de consulta/estudio/referencia necesario para el desarrollo de las tareas
- El fichero `fuentes.zip` contiene todos los ejemplos descritos en `RefZMQ.pdf`

2. Tareas

2.1. Publicador rotatorio

Utilizando el patrón `pub/sub` (difusión) descrito en `RefZMQ.pdf`, desarrolla un programa `publicador.js` que se invoca como

```
node publicador port numMensajes tema1 tema2 ...
```

donde:

- `port` es el port al que deben conectar los subscriptores
- `numMensajes` es el número total de mensajes a emitir (un mensaje por segundo), tras lo cual termina
- `tema1 tema2 ...` es un número variable de temas que se recorren según un turno rotatorio

Ej. tras la invocación `node publicador 8888 5 Politica Futbol Cultura` el publicador debe emitir

- Tras 1 segundo: `1: Politica 1`
- Tras 2 segundos: `2: Futbol 1`
- Tras 3 segundos: `3: Cultura 1`
- Tras 4 segundos: `4: Politica 2`
- Tras 5 segundos: `5: Futbol 2`

O sea, un primer valor correspondiente al segundo, el tema, y un segundo número que corresponde a la cantidad de mensajes de dicho tipo que se han emitido.

2.2. Prueba aplicación chat

En `RefZMQ.pdf` se diseña e implementa una aplicación chat rudimentaria. Analiza el código para comprender su funcionamiento, y comprueba su funcionamiento.

2.3. Parametrización broker

Asumiendo el patrón broker implementado con sockets `router/router` descrito en `RefZMQ.pdf`, modifica el código para aceptar los siguientes parámetros en línea de órdenes:

- `node broker.js portFrontend portBackend`
- n invocaciones `node worker.js urlBackend nickWorker txtRespuesta`
- m invocaciones `node client.js urlFrontend nickClient txtPeticion`

Los valores de los argumentos deben ser coherentes entre sí. Por ejemplo:

- `node broker.js 8000 8001`
- `node worker.js tcp://localhost:8001 W1 Resp1`
- `node worker.js tcp://localhost:8001 W2 Resp2`
- `node client.js tcp://localhost:8000 C1 Hello`
- `node client.js tcp://localhost:8000 C2 Hola`
- `node client.js tcp://localhost:8000 C3 Hi`

Cuando un cliente lanza una petición (ej mensaje "txtPeticion") debe recibir como respuesta "txtRespuesta n", donde n el un valor numérico que indica la cantidad de respuestas que se han generado hasta ese momento entre todos los workers.

Si se quiere hacer una prueba con varios clientes y/o workers, es conveniente tener en cuenta lo siguiente:

- En lugar de abrir un terminal para cada orden, se pueden lanzar varias instancias con una única orden:
  - `node client.js & node client.js & ...`
  - o mediante un shell-script (al que se pasa como argumento la cantidad de instancias)
- En ese caso no resulta tan fácil finalizar la ejecución de cada uno (no tenemos un terminal dedicado donde pulsar ctrl-C)
  - Una opción es usar la orden `kill pid1 pid2 ...`, donde `pidX` es el número de proceso que se observa al lanzar órdenes en background
  - Otra solución es limitar la duración de cada cliente/trabajador introduciendo en el programa la sentencia `setTimeout(()=>{process.exit(0)}, ms)`, donde `ms` indica el tiempo de vida del programa en milisegundos

2.4. Estadísticas broker

Asumiendo el patrón broker implementado con sockets `router/router` descrito en `RefZMQ.pdf`, añade el código necesario en el broker para mantener el total de peticiones atendidas y el número de peticiones atendidas por cada worker. Dicha información debe mostrarse en pantalla cada 5 segundos

2.5. Broker para clientes + Broker para workers

Asumiendo el patrón broker implementado con sockets `router/router` descrito en `RefZMQ.pdf`, se trata de dividir el broker actual en 2 (broker1 y broker2):

- Broker1 se responsabiliza de los clientes, y mantiene la cola de peticiones pendientes
- Broker2 se responsabiliza de los workers (alta, baja, equilibrado de carga, ..)

El funcionamiento es el siguiente:

- Los clientes envían las peticiones a Broker1, que las pasa a Broker2, y éste al worker que corresponda
- La respuesta del worker llega a Broker2, que la pasa a Broker1, y éste al cliente

La solución elegida debe mantener las mismas características externas que observaban workers y clientes

2.6. Prueba del patrón broker tolerante a fallos

`RefZMQ.pdf` describe distintas implementaciones del patrón broker. Compara el funcionamiento del broker `router/router` y el broker tolerante a fallos de los workers.

1. Lanzamos el broker `router/router` y tres procesos worker, eliminamos el primero de los workers mediante la orden `kill`, y lanzamos 3 clientes

- `$ node broker & node worker & node worker & node worker &`
  `[1] 10300 [2] 10301 [3] 10302 [4] 10303`
  `$ kill 10301`
  `$ node client & node client & nodel client &`
- Anotamos cuántas respuestas se obtienen, qué trabajadores las han enviado, y si quedan clientes esperando
- Eliminamos todos los procesos: `killall node`

2. Repetimos la misma prueba, pero utilizando el broker que tolera fallos de los workers