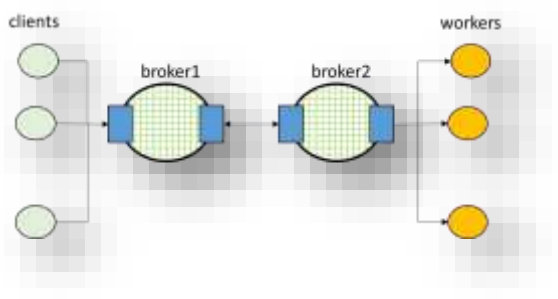
# TSR – EXAMEN LABORATORIO 2, 02-12-2021

**Pregunta 1** (4 puntos) Los dos programas siguientes (**broker1** y **broker2**) representan uno de los posibles intentos de solución al problema de dividir el broker en dos, tal y como se propone en la práctica 2

**broker1.js**

```
01: const zmq = require('zeromq')
02: let nw=0, cli=[], msg=[]
03: let sc = zmq.socket('router')
04: let sb = zmq.socket(*****)
05: sc.bind('tcp://*:9990')
06: sb.bind('tcp://*:9991')
07:
08: function dispatch(c,m) {
09:    nw--
10:    sb.send([c,'',m])
11: }
12:
13: sc.on('message',(c,sep,m) => {
14:    if (nw!=0) dispatch(c,m)
15:    else {cli.push(c); msg.push(m)}
16: })
17:
18: sb.on('message',(c,sep,r) => {
19:    nw++
20:    if (c!='') sc.send([c,'',r])
21:    //
22: })
```

NOTA.- asume que clientes y workers son parametrizables con su id y url donde conectar (por lo demás son los mismos usados en prácticas)




**broker2.js**

```
01: const zmq = require('zeromq')
02: let workers=[]
03: let sb = zmq.socket(*****)
04: let sw = zmq.socket('router')
05: sb.connect('tcp://localhost:9991')
06: sw.bind('tcp://*:9992')
07:
08: sw.on('message', (w,sep,c,sep2,r) => {
09:    workers.push(w)
10:    sb.send([c,'',r])
11:
12: })
13:
14: sb.on('message', (c,sep,m) => {
15:   sw.send([workers.shift(),'',c,'',m])
16: })
```

Responde de forma razonada a las siguientes cuestiones:

a) Indica el tipo de socket que podemos usar para el socket `sb` de broker1 y para el socket `sb` de broker2 (líneas 4 de broker1 y 3 de broker2). Justifica la respuesta.
b) Justifica si el código propuesto funcionará correctamente en aquellas situaciones relacionadas con la disponibilidad de workers.
c) Justifica si debería añadirse código adicional a partir de la línea 21 de broker1 para que el doble broker tuviera un funcionamiento correcto. De ser así, escribe ese código.

---

**Pregunta 2** (1 punto) La versión del **broker con tolerancia a fallos** incluida en la práctica 2 permite experimentar con ciertos escenarios de error. Debes contestar a las siguientes preguntas relacionadas con ese sistema:

a) Explica qué ocurre si el broker falla. Debes mencionar el efecto sobre el resto de componentes, las posibles peticiones en curso y el sistema completo.
b) Explica qué ocurre si falla un worker. Debes incluir las diferencias entre los casos en los que, en el momento del fallo, el worker estuviera procesando una solicitud o a la espera.
c) Explica qué ocurre si un worker, que el broker consideró *averiado*, devuelve con retraso al broker una respuesta `r` a la solicitud `m` de un cliente c.

**Pregunta 3** (4 puntos) Disponemos de un **sistema de chat** idéntico al descrito en el apartado 5 de la práctica 2, del que se muestra el código de un cliente.

```
01: const zmq = require('zeromq')
02: const nick='Ana' //Assume it's random.
03: let sub = zmq.socket('sub')
04: let psh = zmq.socket('push')
05: sub.connect('tcp://127.0.0.1:9998')
06: psh.connect('tcp://127.0.0.1:9999')
07: sub.subscribe('')
08: sub.on('message', (nick,m) => {
09:    console.log('['+nick+']'+m)
10: })
11: process.stdin.resume()
12: process.stdin.setEncoding('utf8')
```

```
13: process.stdin.on('data'  ,(str) => {
14:    psh.send([nick, str.slice(0,-1)])
15: })
16: process.stdin.on('end',() => {
17:    psh.send([nick, 'BYE'])
18:    sub.close(); psh.close()
19: })
20: process.on('SIGINT',() => {
21:    process.stdin.end()
22: })
23: psh.send([nick,'HI'])
```

Uno de los participantes en dicho chat (el "*mafioso*") ha ideado una forma de abusar del sistema por medio de clientes del chat ficticios ("*esbirros*") que obedecen sus órdenes. Su operativa es…

- Cuando el mafioso lee un mensaje del chat (*mensaje_original*) procedente de un cliente concreto (lo denominamos *objetivo*), reaccionará ordenando que cada *esbirro* envíe un mensaje al chat con contenido "**No me gusta el mensaje de *objetivo*: *mensaje_original***".
- Tanto el *mafioso* como los *esbirros* serán versiones modificadas del cliente genérico de chat. Es extremadamente conveniente conocer que **su implementación solo añade instrucciones al original**, manteniendo una separación limpia entre la parte *cliente de chat* y la parte de interacción *mafioso-esbirros*.
    - La única excepción es el momento que provoca la reacción del *mafioso*.
- El *mafioso* y los *esbirros* reciben desde la línea de órdenes, en su variable ***port***, el número de puerto a utilizar para intercomunicarse. El mafioso recibe también en su variable ***target*** el identificador del usuario a molestar. No es necesario escribir código para asignar valor a esas variables en las cuestiones c) y d).

Diseña el código de mafioso y esbirros para contestar a las siguientes cuestiones:

a) Elige el/los tipo/s de socket/s ZeroMQ para comunicar mafioso y esbirros. Argumenta la elección.
b) El código del mafioso se basa en el cliente de chat. ¿Qué instrucciones insertarías en ese código para detectar el mensaje del objetivo e iniciar la reacción?. Indica dónde referenciando el/os números de línea del código del cliente de chat.
c) Siguiendo con el *mafioso*, escribe las instrucciones que añadirías al código del cliente de chat para construir el programa `mafioso`.
d) Centrándonos en los *esbirros*, escribe las instrucciones que añadirías al código del cliente de chat para construir el programa `esbirro`.

---

**Pregunta 4** (1 punto) En uno de los apartados de la práctica 2 se pide visualizar periódicamente **estadísticas sobre las peticiones atendidas**, pero en esta pregunta no nos interesa el número total. Explica qué estructuras de datos has diseñado para mantener la información necesaria **para cada worker**, y cómo accedes a las mismas. Ilústralo implementando la función que, según el enunciado, debería mostrar esa información cada 5 segundos en pantalla (por ejemplo, una función `visualize()` invocada mediante `setInterval(visualize, 5000)`)