Esta prueba consta de 6 cuestiones tipo test (0.6 puntos), 2 cuestiones de respuesta breve (0.4 puntos), y una pregunta de respuesta abierta (1 punto). En el test cada error descuenta 1/3 de un acierto. El test se responde en el propio enunciado, el resto en papel aparte. **Indica nombre y grupo**

Suponemos esta implementación del broker tolerante a fallos (FT)

```
1   const zmq = require('zeromq');
2   const ansInterval = 2000;
3   let who=[], req=[], workers=[], tout={}
4
5   let sc = zmq.socket('router') // frontend
6   let sw = zmq.socket('router') // backend
7   sc.bind('tcp://*:9998')
8   sw.bind('tcp://*:9999')
9
10  function resend(c, m) {
11    return function () {
12      sw.send([workers.shift(), '', c, '', m])
13    }
14  }
15
16  function sendToW(w, c, m) {
17    sw.send([w, '', c, '', m])
18    tout[w] = setTimeout(resend(c,m),
19      ansInterval)
20  }
21
22  sc.on('message', (c, sep, m) => {
23    if (workers.length == 0) {
24      who.push(c); req.push(m)
25    } else {
26      sendToW(workers.shift(),c, m)
27    }
28  });
29
30  sw.on('message', (w, sep, c, sep2, r) => {
31    if (c == '') {workers.push(w); return}
32    clearTimeout(tout[w]); delete tout[w]
33    if (who.length == 0) {
34      workers.push(w)
35    } else {
36      sendToW (w, who.shift(), req.shift())
37    }
38    sc.send([c,'',r])
39  })
```

1 *En relación con el código del broker FT, la función* `resend(c, m)`

- **a** Envía el mensaje `m` al cliente `c`
- **b** Selecciona el primer worker en espera, y le envía el trabajo `m`
- **c** Crea una clausura que al ejecutarse, siempre selecciona un worker, y le envía el trabajo `m`
- **d** Crea una clausura que al ejecutarse por el broker, puede fallar al no encontrar un worker disponible.

2 *En relación con el código del broker FT, en el array* `who` …

- **a** Aparecen solo identidades de workers
- **b** Aparecen tareas enviadas por los clientes
- **c** Aparecen solo identidades de clientes
- **d** Aparecen identificadores de clientes y de workers

3 *En relación con el código del broker FT, en el array* `who` …

- **a** Puede repetirse la identidad de un cliente
- **b** Puede repetirse la identidad de un worker
- **c** Puede repetirse mensajes de los clientes
- **d** Nunca aparecen elementos repetidos

4 *En relación con el código del broker FT, el array* `req` …

- **a** No está relacionado con el array `who`
- **b** Contiene mensajes de los clientes
- **c** Contiene respuestas de los workers
- **d** Contiene identificadores de temporizaciones

5 *En relación con el código del broker FT, el manejador de mensajes de los workers*

- **a** Siempre almacena al worker en la lista de workers disponibles
- **b** Sólo almacena a un worker en `workers` si no hay trabajos de clientes esperando ser servidos
- **c** Siempre envía un mensaje de respuesta a un cliente
- **d** Ninguna de las otras afirmaciones es cierta

6 *En relación con el código del broker FT, el broker…*

- **a** Cesa su ejecución cuando un worker falla.
- **b** Asume fallo de worker cuando dicho worker tarda en responder más de `ansInterval` ms
- **c** Mantiene la lista de workers que han fallado
- **d** Avisa a un cliente cuando un worker falla

Cuestión 1.- 0.2 puntos

Suponemos un *publicador rotatorio* similar al desarrollado en la práctica, pero que en lugar de leer los valores numMensajes y lista de temas desde línea de órdenes, los recibe desde un agente externo a través de un socket.

Define el tipo de socket, formato de mensaje, y manejador para dichos mensajes

Cuestión 2.- 0.2 puntos

Describe los sockets y formato de los mensajes a utilizar para resolver el apartado *Broker para clientes + Broker para workers* de la práctica.

Con esa estructura, y en caso de que el broker para clientes reciba dos peticiones p1,p2 en ese orden, ¿es posible que la respuesta a la petición p2 llegue a dicho broker antes que la respuesta a p1?

Ejercicio de respuesta abierta.- 1 punto

El código del broker FT proporcionado es muy simple, y son posibles distintas mejoras. Plantea las modificaciones al código necesarias para resolver los siguientes aspectos:

1 El código actual asume que arrancamos los workers antes de arrancar los clientes. Modifica el código para contemplar la posibilidad de que cuando arranca un worker ya existan peticiones pendientes

2 Cuando vence el timeout se reenvía la petición a otro worker. Modifica el código para contemplar el caso en que tras vencer el timeout no hay workers disponibles

3 El código actual no contempla la posibilidad de una respuesta tardía (respuesta tras vencer el timeout, o sea de un nodo que pensábamos que había fallado). Modifica el código para garantizar que dichas respuestas tardías se ignoran

Soluciones al test

dcdbdb

Soluciones al ejercicio de respuesta abierta

apartado 1)

Una posible solución es cambiar línea 31 por

```
if (c==''){
    if (who.length==0) workers.push()
    else sendToW(w,who.shift(),req.shift())
    return
}
```

apartado 2)

Una posible solución es reescribir resend

```
function resend(c,m){
    return function(){
        if (workers.length==0){
            who.push(c); req.push(m)
        }
        else sw.send(workers.shift(),'',c,'',m)
    }
}
```

apartado 3)

Son posibles distintas alternativas de solución

1) Añadir la lista de workers que han fallado, y descartar sus mensajes

```
let wfail={}
wfail[w]=true; // entre 11 y 12
if (!wfail[w]) sc.send([c,'',r]) // linea 38
```

En ese caso hay que modificar la 31 (recup. de nodo)

```
if (c==''){
    if (wfail[w]) delete wfail[w]
    workers.push(2); return
}
```

2) Añadir la lista de clientes esperando respuesta. En este caso sólo garantiza respuesta única si entre la primera y la segunda respuesta no se ha generado otra petición del mismo cliente; además no cumple estrictamente el enunciado (descartar la respuesta fuera de plazo), porque puede que esté descartando la otra

```
let cli={}      // linea 4
cli[c]=true;    // entre líneas 22 y 23. c esta pendiente de respuesta
if (cli[c]) {sc.send([c,'',r]); delete cli[c]} // linea 38. La primera resp elimina cli[c], y la segunda no se envia
```