TSR: Solución Recuperación Práctica 2

```
 1: // clienteChat.js
 2: const {zmq, lineaOrdenes, error, adios,
 3:    conecta} = require('../tsr')
 4: lineaOrdenes("nick hostServidor " +
 5:    "portDifusion portPipeline")
 6: // FQDN or IP address of this computer. To
 7: // be used in URLs, if needed.
 8: const hostname = require('os').hostname()
 9:
10: let entrada = zmq.socket('sub')
11: let salida  = zmq.socket('push')
12: conecta(salida, hostServidor, portPipeline)
13: conecta(entrada,hostServidor, portDifusion)
14: entrada.subscribe('')
15: entrada.on('message', (nick,m) =>
16:       {console.log('['+nick+'] '+m)})
17:
18: process.stdin.resume()
19: process.stdin.setEncoding('utf8')
20: process.stdin.on('data', (str)=>
21:     {salida.send([nick, str.slice(0,-1)])})
22: process.stdin.on('end',()=>
23:     {salida.send([nick, 'BYE']);
24:      adios([entrada,salida],"Adios")()})
25:
26: salida.on('error', (msg) =>
27:     {error(`${msg}`)})
28: process.on('SIGINT',adios([entrada,salida],
29:     "abortado con CTRL-C"))
30:
31: salida.send([nick,'HI'])
```

```
 1: // servidorChat.js
 2: const {zmq, lineaOrdenes, error, adios,
 3:   creaPuntoConexion} = require('../tsr')
 4: lineaOrdenes("portDifusion portEntrada")
 5:
 6: let entrada = zmq.socket('pull')
 7: let salida  = zmq.socket('pub')
 8: creaPuntoConexion(salida, portDifusion)
 9: creaPuntoConexion(entrada,portEntrada)
10:
11: entrada.on('message', (id,txt) => {
12:   switch (txt.toString()) {
13:     case 'HI':
14:       salida.send(['server',id+' connected'])
15:       break
16:     case 'BYE':
17:       salida.send(['server', id+
18:         ' disconnected'])
19:       break
20:     default:
21:       salida.send([id,txt])
22:   }
23: })
24:
25: salida.on('error', (msg) => {error(`${msg}`)})
26: process.on('SIGINT', adios([entrada,salida],
27:   "abortado con CTRL-C"))
28:
29:
30:
31:
```

Se pretende añadir al **chat** un *control de identidad*. Para ello, en el mensaje de anuncio ("HI") se añadirá un segmento conteniendo una cadena (key) además del nick, de manera que el formato pase a ser [nick, key, contenido] en todos los mensajes enviados por los clientes. Las condiciones de uso son:

- El mensaje HI solo es admitido si el nick no existe. En caso positivo, asocia la key a dicho nick (lo registra).
- Los mensajes posteriores solo son admitidos si el par nick y key del mensaje está registrado.
- El mensaje BYE tiene el mismo requisito anterior. En caso positivo, elimina la asociación entre nick y key.

CUESTIÓN 1 (4 puntos)

Se solicita la implementación del **servidor** de chat en esta nueva situación, sin devolver ningún mensaje de respuesta cuando el par nick y key del cliente no son correctos (mismo efecto que si esa solicitud no hubiera llegado). No hay nuevos sockets ni se debe implementar el código del cliente.

CUESTIÓN 2 (6 puntos)

Se solicita incorporar algún mecanismo (como pueda ser un nuevo socket de respuesta en los clientes, o el uso de segmentos adicionales en algunos mensajes), que permita comunicación **asincrónica** para que el cliente reciba las notificaciones de error por el mensaje inicial de anuncio. El contenido será el original que el cliente envió. Los clientes no pueden recibir mensajes destinados a otros. Debe argumentarse…

a) qué sockets asincrónicos colocar en el servidor y en los clientes, si se añaden o sustituyen alguno de los originales,
b) cómo interactúan,
c) cómo se garantiza que la notificación de error llega exclusivamente al cliente que invocó el anuncio ("HI") con parámetros incorrectos.

SOLUCIÓN CUESTIÓN 1

Aspectos importantes:

1. Debe diseñarse una estructura de datos que recuerde las claves (key) registradas por cada cliente (id). Ese tipo de construcciones, basadas en diccionarios, son bastante sencillas en JavaScript, y las hemos aprovechado en el laboratorio 2.
2. Se añaden y retiran entradas del diccionario keys cuando llegan mensajes 'HI'y 'BYE'.
3. Se comprueban entradas en todos los mensajes que llegan al servidor.
4. NO se altera nada más, salvo añadir un nuevo segmento al mensaje proporcionado por el cliente

Vamos allá con el nuevo código del servidor, con fondo gris para destacar cambios…

```
// sol1_servidorChat.js
const {zmq, lineaOrdenes, error, adios,
  creaPuntoConexion} = require('../tsr')
lineaOrdenes("portDifusion portEntrada")

var keys=[]

let entrada = zmq.socket('pull')
let salida  = zmq.socket('pub')
creaPuntoConexion(salida, portDifusion)
creaPuntoConexion(entrada,portEntrada)

entrada.on('message', (id,key,txt) => {
  switch (txt.toString()) {
    case 'HI':
      if (keys[id] == undefined) {
        keys[id] = key
        salida.send(['server',id+' connected'])
      }
      break
    case 'BYE':
        if (keys[id] == key) {
            salida.send(['server', id+
        ' disconnected'])
            delete(keys[id])
        }
      break
    default:
        if (keys[id] == key)
           salida.send([id,txt])
  }
})
salida.on('error', (msg) => {error(`${msg}`)})
process.on('SIGINT', adios([entrada,salida], "abortado con CTRL-C"))
```

SOLUCIÓN CUESTIÓN 2

La cuestión anterior servía como paso intermedio para resolver el nuevo escenario, pero carece de notificación de los posibles errores a los clientes. El enunciado de la cuestión 2 informa que solo se desea notificar en caso de error por el mensaje inicial de anuncio. Esto supone que hemos de separar el tratamiento del mensaje de anuncio (`'HI'`). Por otro lado se añade la necesidad de contestar al cliente, aspecto extremadamente difícil de cubrir con los sockets PUB/SUB y PUSH/PULL actuales, y que nos lleva a añadir un socket adicional para que los clientes envíen mensajes de anuncio y reciban, en su caso, respuestas asincrónicas.

La elección de nuevos sockets asincrónicos y su interacción son los elementos centrales del ejercicio. Se plantean algunas alternativas básicas, pero pueden aceptarse otras que también respeten las restricciones establecidas en esta cuestión: ampliar el sistema PUSH/PULL con otro par (PULL/PUSH) para proporcionar bidireccionalidad, añadir un par DEALER/ROUTER exclusivamente para los mensajes de anuncio, sustituir todos los sockets originales por un ROUTER en el servidor y un DEALER en el cliente…

1. En la primera alternativa el servidor podría contar con un socket PUSH por cada cliente para enviar esas respuestas y los clientes deberían tener un socket PULL para recibirla. Como el problema inicial no se limita a un número determinado de clientes, se adoptará la estrategia de un único socket PUSH que conectará/desconectará del cliente que realice `'HI'`. Para desconectar el socket, este puede cerrarse con close() y volverse a crear con zmq.socket(), dejándolo así preparado para el connect() a realizar con el siguiente HI. El formato del mensaje HI del cliente debe contener otro segmento con la URL a la que conectará y contestará el servidor solo para ese HI (aprovechamos "`hostname`" para ello, y un puerto predeterminado).
    - No es equivalente a un único DEALER para que el servidor conteste y al que los PULL de los clientes conecten porque eso NO aseguraría que las respuestas lleguen al cliente concreto.
2. En la segunda alternativa, cada cliente tendrá un nuevo socket DEALER que utilizará para enviar el mensaje HI y su hipotética respuesta, y el servidor añade un socket ROUTER, de manera que recibe HI por una cola y contesta por la misma. No hay forma de que otro cliente reciba esa respuesta. Para que sea equivalente al caso inicial, los mensajes contestados positivamente deberán también copiarse al canal PUB.
3. En la tercera alternativa el servidor sustituiría sus sockets PUB y PULL por un único socket ROUTER, mientras los clientes sustituirían sus sockets SUB y PUSH por un único socket DEALER. El servidor deberá recordar las identidades de los clientes, obtenidas automáticamente en el primer segmento del mensaje HI enviado por cada cliente. De esa manera podrá tanto difundir mensajes a todos los clientes (con un bucle en el que se recorran todas las identidades registradas hasta el momento) como enviar una respuesta a un cliente específico (usando un solo mensaje con su identidad en el primer segmento, con lo que se asegura que los demás no la recibirán). Los sockets DEALER de los clientes son bidireccionales y asincrónicos, por lo que admiten la comunicación tal como exige esta cuestión.

Aunque el enunciado no lo requiere, a continuación se proporciona una **implementación de la segunda alternativa** (DEALER/ROUTER).

El código del nuevo cliente incorpora un nuevo socket `portHI` del tipo DEALER, para conectar con el ROUTER del servidor, y una clave que se suministrará en todos los mensajes como segundo argumento. Dispone de un

listener asociado al mismo socket para las posibles respuestas del servidor, que serán siempre notificaciones de error. Como tareas complementarias, se necesita eliminar el nuevo socket al finalizar el proceso.

```
// sol2_clienteChat.js
const {zmq, lineaOrdenes, error, adios,
   conecta} = require('../tsr')
lineaOrdenes("nick key hostServidor " +
   "portDifusion portPipeline portHI")
// FQDN or IP address of this computer. To
// be used in URLs, if needed.
const hostname = require('os').hostname()

// The key is received as a command-line argument.
let soHI = zmq.socket('dealer')
conecta(soHI, hostServidor, portHI)
soHI.send([nick,key,'HI'])
soHI.on('message', (id,key,txt) => {
    console.log("Problems... %s, %s, %s", id, key, txt)
})

let entrada = zmq.socket('sub')
let salida  = zmq.socket('push')
conecta(salida, hostServidor, portPipeline)
conecta(entrada,hostServidor, portDifusion)
entrada.subscribe('')
entrada.on('message', (nick,m) =>
      {console.log('['+nick+'] '+m)})

process.stdin.resume()
process.stdin.setEncoding('utf8')
process.stdin.on('data', (str)=>
    {salida.send([nick, key, str.slice(0,-1)])})
process.stdin.on('end',()=>
    {salida.send([nick, key, 'BYE']);
     adios([entrada,salida,soHI],"Adios")()})

salida.on('error', (msg) =>
    {error(`${msg}`)})
process.on('SIGINT',adios([entrada,salida,soHI], "abortado con CTRL-C"))
```

El código del nuevo servidor incorpora un nuevo socket `portHI` del tipo ROUTER, para conectar con los DEALER de cada cliente. Sabemos que se crea una cola para cada cliente, de forma que el servidor puede dirigir la notificación de error al cliente que está atendiendo.

Puede mencionarse que el *listener* asociado al socket ROUTER solo recibe mensajes de anuncio (se ha obviado comprobarlo), de manera que éstos ya no son considerados en el *listener* del otro socket. Como tareas complementarias, se necesita eliminar el nuevo socket al finalizar el proceso.

```
// sol2_servidorChat.js
const {zmq, lineaOrdenes, error, adios,
  creaPuntoConexion} = require('../tsr')
lineaOrdenes("portDifusion portEntrada portHI")

var keys=[]
let soHI  = zmq.socket('router')
creaPuntoConexion(soHI, portHI)
soHI.on('message', (q,id,key,txt) => {
    id += ''; key += ''
    if (keys[id] == undefined) {
      keys[id] = key
      salida.send(['server',id+' connected'])
    }
    else soHI.send([q,id,key,txt])
})

let entrada = zmq.socket('pull')
let salida  = zmq.socket('pub')
creaPuntoConexion(salida, portDifusion)
creaPuntoConexion(entrada,portEntrada)

entrada.on('message', (id,key,txt) => {
  switch (txt.toString()) {
  // el tipo HI es atendido mediante soHI
    case 'BYE':
        if (keys[id] == key) {
            salida.send(['server', id+
        ' disconnected'])
            delete(keys[id])
        }
      break
    default:
        if (keys[id] == key)
           salida.send([id,txt])
  }
})

salida.on('error', (msg) => {error(`${msg}`)})
process.on('SIGINT', adios([entrada,salida,soHI], "abortado con CTRL-C"))
```