TSR

Examen de las sesiones 1 y 2 de la Práctica 2

1. Dado el código del fichero origen1.js de la práctica 2...

```
 1: const {zmq, lineaOrdenes, error, adios, conecta} = require('../tsr')
 2: lineaOrdenes("nombre hostSig portSig")
 3: let salida = zmq.socket('push')
 4: conecta(salida, hostSig, portSig)
 5: salida.on('error', (msg) => {error(`${msg}`)})
 6: process.on('SIGINT', adios([salida],"abortado con CTRL-C"))
 7: for (let i=1; i<=4; i++) {
 8:         console.log(`enviando mensaje: [${nombre},${i}]`)
 9:         salida.send([nombre,i])
10: }
```

Y el código del fichero destino.js:

```
 1: const {zmq, error, lineaOrdenes, traza, adios, creaPuntoConexion} = require('../tsr')
 2: lineaOrdenes("nombre port")
 3: let entrada = zmq.socket('pull')
 4: creaPuntoConexion(entrada, port)
 5: function procesaMensaje(filtro, nombre, iteracion) {
 6:         if (!iteracion) {
 7:                 iteracion = nombre
 8:                 nombre = filtro
 9:                 filtro = ""
10:         }
11:         traza('procesaMensaje','filtro nombre iteracion',[filtro, nombre, iteracion])
12: }
13: entrada.on('message', procesaMensaje)
14: entrada.on('error', (msg) => {error(`${msg}`)})
15: process.on('SIGINT', adios([entrada],"abortado con CTRL-C"))
```

Se solicita escribir el programa *filtro.js*, en el que no será necesario gestionar los posibles eventos 'error', y adaptar el programa *origen1.js* de manera que respeten todas estas condiciones simultáneamente:

- **Un solo proceso que ejecute el programa *origen1.js* adaptado podrá interactuar con tantos procesos que ejecuten *filtro.js* como decida el usuario. (15%)**
- **El programa *origen1.js* adaptado no debe enviar cuatro mensajes, sino un mensaje cada segundo, mientras dure su ejecución. El primer mensaje incluirá un 1 como valor de su segundo segmento. Cada nuevo envío incrementará ese valor en una unidad. (20%)**
- **El programa *filtro.js* recibirá desde la línea de órdenes el nombre de ese filtro (el usuario podrá iniciar varios, cada uno con un nombre diferente), así como la información necesaria para intercomunicarse con *origen1.js* y *destino.js*. (15%)**
- **Al filtrar cada mensaje recibido, el programa *filtro.js* incluirá su nombre como primer segmento adicional y duplicará el valor recibido en el último segmento. (15%)**
- **El programa *filtro.js* debe utilizar los tipos de socket adecuados y las operaciones necesarias para que la comunicación sea posible entre los tres tipos de proceso. (15%)**
- **El programa *filtro.js* espera al menos el número de milisegundos especificado en el último segmento del mensaje recibido antes de reenviar ese mensaje filtrado a *destino.js*. (20%)**

TSR

Examen de las sesiones 1 y 2 de la Práctica 2

SOLUCIÓN

Esta preguna admite múltiples soluciones. Hay varias formas de desarrollar el nuevo programa origen de manera que cumpla todos los requisitos mencionados en el enunciado.

Aunque no es imprescindible, podemos caer en la cuenta de que al describir el uso de *connect* o *bind* en el Tema 3 se indicó que sería recomendable utilizar *bind* en aquellos casos en que el URL correspondiente pudiera ser utilizado por múltiples procesos remotos para establecer conexiones con él. Eso es lo que está sucediendo en este ejemplo particular. Por tanto, atendiendo a ese hecho, el programa origen podría haber sido el siguiente:

```
const {zmq, lineaOrdenes, error, adios, creaPuntoConexion} = require('../tsr')
lineaOrdenes("nombre portSig")
let salida = zmq.socket('push')
creaPuntoConexion(salida, portSig)
salida.on('error', (msg) => {error(`${msg}`)})
process.on('SIGINT', adios([salida],"abortado con CTRL-C"))
let contador = 1
setInterval(()=>{salida.send([nombre,contador++])}, 1000)
```

De esa manera el programa origen solo necesitará recibir su nombre y el número de puerto desde la línea de órdenes. Con esa información creará el punto de conexión utilizando el socket de tipo PUSH ya declarado en el código original. Posteriormente, habrá que cambiar el bloque final donde se realizaban los cuatro envíos por las dos últimas líneas del código que acabamos de presentar, para enviar un mensaje cada segundo con la información solicitada.

El filtro a desarrollar resultará bastante más complejo, pues ahora tendrá que conectarse tanto con el origen como con el destino. Para ello necesitará ahora cinco argumentos desde la línea de órdenes: su nombre, y tanto el nombre o dirección IP del nodo como el número de puerto, para origen (**Ant**erior en nuestros argumentos) y destino (o **Sig**uiente). Con este diseño, el usuario podrá lanzar tantos filtros como quiera y nuestro origen servirá para proporcionar información a todos ellos, de manera rotatoria, como ya vimos en el ejemplo de "origen2.js" visto en la Práctica 2, que ilustraba esa característica de los sockets PUSH. Además, ahora el usuario tiene total libertad para iniciar esos filtros cuando crea conveniente. No es necesario, iniciarlos todos a la vez.

La gestión de las pausas puede resolverse mediante *setTimeout*. El resultado final se muestra seguidamente:

```
const {zmq, lineaOrdenes, error, adios, conecta} = require('../tsr')
lineaOrdenes("nombre hostAnt portAnt hostSig portSig")
let salida = zmq.socket('push')
let entrada = zmq.socket('pull')
conecta(entrada, hostAnt, portAnt)
conecta(salida, hostSig, portSig)
function procesaMensaje(origen,valor) {
        setTimeout( ()=>{salida.send([nombre,origen,valor*2])}, parseInt(valor+""))
}
entrada.on('message', procesaMensaje)
salida.on('error', (msg) => {error(`${msg}`)}) // Innecesario.
process.on('SIGINT', adios([salida],"abortado con CTRL-C"))
```

TSR

Examen de las sesiones 1 y 2 de la Práctica 2

Otras soluciones válidas podrían utilizar un filtro que hiciera un *bind* en su socket receptor y un *connect* en su socket emisor. Eso implicaría que el nuevo programa origen tendría que recibir toda la información necesaria para conectarse con todos los filtros previstos por el usuario. Eso no incumple lo requerido por el enunciado, pues este no indicaba cuándo debían iniciarse los filtros y, por tanto, resulta aceptable que el número máximo de filtros a utilizar estuviese implícito en la información proporcionada durante el inicio del programa origen.