Apellidos y nombre	

TSR.- Práctica 2 2019-dic-02

Duración 90 minutos. No quite la grapa. Se entrega todo el bloque

ACTIVIDAD 1 (4 puntos)

Considere el siguiente programa, utilizado para resolver parcialmente la actividad propuesta en el apartado 2.1 (Publicador rotatorio) de la Práctica 2:

```
// publisher.js
const zmq = require('zeromq')
let pub = zmq.socket('pub')
let port = process.argv[2] || 56621
let nmsg = parseInt(process.argv[3]) || 5
let topics = process.argv.slice(4)
if (!topics[0])
    topics = ['t1', 't2', 't3']
let counter = 0
pub.bindSync('tcp://*:'+port)
function emit() {
        let m=topics[0]
        pub.send(++counter + ': ' + m)
        // Circular management
        topics.shift(); topics.push(m)
        if (counter==nmsg)
                process.exit(0)
}
setInterval(emit,1000)
```

Este publicador no llega a manejar el segundo contador mencionado en el enunciado, pero no importa.

Escriba el programa suscriptor necesario para recibir todos los mensajes difundidos por este publicador y mostrarlos por pantalla. Asuma que este suscriptor siempre se utilizará en el mismo ordenador en el que se inicie el publicador y que se iniciarán todos los suscriptores antes que el publicador. El suscriptor deberá recibir desde la línea de órdenes el número de puerto al que deberá conectarse. Si no se facilitara ningún argumento, debería ser capaz de conectarse correctamente a un publicador que no reciba argumentos.

ACTIVIDAD 2 (6 puntos)

Considere los siguientes programas, derivados del broker ROUTER/ROUTER presentado en el documento RefZMQ utilizado en la Práctica 2:

```
// File: broker.js
const zmq = require('zeromq')
let cli=[], req=[], workers=[]
let sc = zmq.socket('router')
let sw = zmq.socket('router')
sc.bind('tcp://*:9998')
sw.bind('tcp://*:9999')
sc.on('message',(c,sep,m)=> {
 if (workers.length==0) {
  cli.push(c); req.push(m)
 } else
  sw.send([workers.shift(),'',
   c,'',m])
})
sw.on('message',(w,sep,c,sep2,r)=> {
 if (c=='') {workers.push(w);
  return}
 if (cli.length>0) {
  sw.send([w,'',
  cli.shift(),'',req.shift()])
 } else {
  workers.push(w)
 }
 sc.send([c,'',r])
})
```

```
// File: client.js
const zmq = require('zeromq')
let req = zmq.socket('req')
let who = process.argv[2]
req.connect('tcp://localhost:9998')
req.on('message', (msg)=> {
 console.log(msg+'')
 process.exit(0)
})
req.send(who)
```

```
// File: worker.js
const zmq = require('zeromq')
let req = zmq.socket('req')
let id = process.argv[2]
req.identity=id
req.connect('tcp://localhost:9999')
req.on('message', (c,sep,msg)=> {
 setTimeout(()=> {
  req.send([c,'',msg+' '+id])
 }, 1000)
})
req.send(['','',''])
```

En estos ficheros se han realizado algunas modificaciones (subrayadas para que sea fácil identificarlas) con el objetivo de distinguir qué cliente y trabajador intervienen en la gestión de los mensajes.

Amplíe el programa broker.js para que, al igual que en el apartado 2.4 de la práctica 2, recoja las estadísticas de uso siguientes:

- **Total de peticiones atendidas hasta el momento por todos los trabajadores.**
- **Número de peticiones atendidas hasta el momento por cada worker.**

Sin embargo, en lugar de mostrar esa información cada cinco segundos, este broker añadirá toda esa información, concatenándola a las respuestas redirigidas a los clientes. Utilice para ello el carácter '\n' como separador de líneas.

Por ejemplo, si la respuesta debía ser inicialmente: "cli1 wor3", ahora la respuesta mostrada por ese cliente podría ser:

```
cli1 wor3
Total peticiones: 4
wor1: 2 peticiones
wor2: 1 peticiones
wor3: 1 peticiones
```

ACTIVIDAD 1: SOLUCIÓN

Una posible solución sería:

```
// subscriber.js
const zmq = require('zeromq')
let sub = zmq.socket('sub')
let port = process.argv[2] || 56621
sub.connect('tcp://127.0.0.1:'+port)
sub.subscribe('')
sub.on('message', (msg) => {console.log(''+msg)})
```

En ella, debe empezarse importando el módulo 'zeromq'.
El socket únicamente puede ser de tipo SUB.
El puerto a utilizar se recoge desde la componente 2 del vector process.argv. En caso de que no se haya facilitado ningún valor, se utiliza el operador lógico '||' (O) para que se utilice el número de puerto 56621 (el mismo que utilizó el publicador) por omisión.
El argumento de connect() debe utilizar TCP como su protocolo y la dirección local (127.0.0.1 de la interfaz de *loopback* o, alternativamente, el nombre 'localhost') en sus primeras dos componentes. A ellas se debe concatenar el puerto recibido. No es válido el carácter '*' como dirección, pues un proceso puede conectarse a cualquier ordenador accesible a través de la red. El asterisco solo tiene sentido en la operación bind() o bindSync() pues en ese caso la dirección siempre será local.
El programa necesita utilizar la operación 'subscribe()' del socket y su argumento debe ser una cadena vacía. De otra manera, el suscriptor no podría recibir todos los mensajes emitidos por el publicador.
Finalmente, la gestión del evento 'message' necesitará un *callback* con un único argumento pues el emisor solo envía una cadena en su mensaje y esta ocupa un solo segmento. Como código de ese *callback* se necesitará un console.log() que escriba ese segmento. Por omisión los segmentos recibidos son objetos de la clase Buffer. Para que el contenido se muestre correctamente en la pantalla habrá que convertir previamente ese Buffer en una cadena. Para ello se puede utilizar la operación "toString()" o bien concatenar el Buffer a una cadena vacía.

ACTIVIDAD 2: SOLUCIÓN

Una posible solución sería:

```
// File: broker.js
const zmq = require('zeromq')
let cli=[], req=[], workers=[]
let sc = zmq.socket('router')
let sw = zmq.socket('router')
let total = 0
let countersW = []
sc.bind('tcp://*:9998')
sw.bind('tcp://*:9999')
sc.on('message',(c,sep,m)=> {
        if (workers.length==0) {
                cli.push(c); req.push(m)
        } else
                sw.send([workers.shift(),'',c,'',m])
        })
sw.on('message',(w,sep,c,sep2,r)=> {
        let stats=r +'\nTotal requests: '
        if (c=='') {workers.push(w);
                countersW[w]=0
                return}
        if (cli.length>0) {
                sw.send([w,'',cli.shift(),'',req.shift()])
        } else {
                workers.push(w)
        }
        total++
        countersW[w]++
        stats += total+'\n'
        for (let i in countersW)
                stats += i + ': ' + countersW[i] + ' requests\n'
        sc.send([c,'',stats])
})
```

En ella se han resaltado las líneas que extienden el programa original.