TSR

RECUPERACIÓN DE LA PRÁCTICA 2 (Ejercicio)

Sobre la base del código Br R-R (mostrado abajo) se solicita implementar las siguientes variaciones:

1. Cada worker posee una capacidad inicial de trabajos asignados. Cada vez que sirve una petición, el worker disminuye su contador; el proceso termina cuando el contador llega a 0.
    - La invocación de los workers sigue el formato `node worker.js ID URL N`, donde ID es la identidad del worker, URL permite conectar con el backend, y el nuevo argumento N es la capacidad del worker.
2. El broker gestiona la cola de workers disponibles teniendo en cuenta si siguen teniendo capacidad para atender peticiones (un worker no debe aparecer en workers si su capacidad restante es 0)
    - Entre los workers disponibles se utiliza Round-Robin

Debes indicar el nuevo código de worker y broker. En lugar de escribir el código completo, puedes indicar únicamente los cambios respecto al código proporcionado.

```
 1: // broker
 2: const zmq = require('zmq')
 3: let cli=[], req=[], workers=[]
 4: let sc = zmq.socket('router') // frontend
 5: let sw = zmq.socket('router') // backend
 6: sc.bind('tcp://*:9998')
 7: sw.bind('tcp://*:9999')
 8: sc.on('message',(c,sep,m)=> {
 9:     if (workers.length==0) {
10:         cli.push(c); req.push(m)
11:     } else {
12:         sw.send([workers.shift(),'',c,'',m])
13:     }
14: })
15: sw.on('message',(w,sep,c,sep2,r)=> {
16:     if (c=='') {workers.push(w); return}
17:     if (cli.length>0) {
18:         sw.send([w,'',
19:             cli.shift(),'',req.shift()])
20:     } else {
21:         workers.push(w)
22:     }
23:     sc.send([c,'',r])
24: })
```

```
 1: // worker
 2: const zmq = require('zmq')
 3: let req = zmq.socket('req')
 4: req.identity='Worker1'
 5: req.connect('tcp://localhost:9999')
 6: req.on('message', (c,sep,msg)=> {
 7:     setTimeout(()=> {
 8:         req.send([c,'','resp'])
 9:     }, 1000)
10: })
11: req.send(['','',''])
```

SOLUCIÓN

Los dos apartados que aparecen en el enunciado hacen referencia a los dos componentes que deben extenderse. El "worker" en el primer apartado y el "broker" en el segundo. Empecemos, por tanto, con el componente "worker". Una posible solución sería (hay muchas otras, pues la gestión de la capacidad de un "worker" admite múltiples variantes):

```
// worker
const zmq = require('zmq')
let req = zmq.socket('req')
if (process.argv.length != 5) {
   console.log("These arguments are needed: workerID URL capacity")
   process.exit(-1)
}
req.identity=process.argv[2]
req.connect(process.argv[3])
let counter=parseInt(process.argv[4]+'')
// Its capacity should be a positive value. Otherwise, abort it.
if (counter<1)
        process.exit(-1)
req.on('message', (c,sep,msg)=> {
   setTimeout(()=> {
      counter--
      req.send([c,'','resp',counter])
      if (counter==0)
         process.exit(-1)
   }, 1000)
})
req.send(['','','',counter])
```

En este programa tendremos que iniciar nuestras extensiones comprobando que se han facilitado tres argumentos desde la línea de órdenes. El primero (process.argv[2]) proporciona la identidad del "worker". Esa identidad debe asignarse a la propiedad "identity" del *socket* utilizado por el "worker". El segundo argumento es la URL sobre la que se realizará la conexión. Por último, el tercer argumento especifica la capacidad que tendrá este proceso, en cuanto al número de solicitudes que deberá atender antes de finalizar su ejecución. Guardamos ese valor en una variable "counter". Podemos comprobar en ese momento (aunque no es obligatorio hacerlo) si se ha recibido un valor superior a cero. Si no lo fuera, podríamos abortar aquí la ejecución del trabajador, pues no será capaz de atender ninguna petición y no vale la pena que se registre en el "broker".

El contador de peticiones que todavía pueden atenderse deberá incluirse tanto en el mensaje de registro del "worker" (última línea) como en las respuestas que retorne al "broker". Además, cada vez que se atienda una petición, el valor de ese contador deberá decrementarse. Una vez se envíe la respuesta a la última petición que podía servirse se tendrá que realizar un "process.exit()" para finalizar su ejecución.

Por su parte, las modificaciones en el "broker" generarían un programa similar al siguiente (de nuevo, serían admisibles otras variantes, dependiendo de cómo se haya escrito el "worker"):

```
// broker
const zmq = require('zmq')
let cli=[], req=[], workers=[]
let sc = zmq.socket('router') // frontend
let sw = zmq.socket('router') // backend
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,cap)=> {
   if (c=='') {workers.push(w); return}
   let capacity=parseInt(cap+'')
   if (capacity>0)
      if (cli.length>0) {
         sw.send([w,'',cli.shift(),'',req.shift()])
      } else {
         workers.push(w)
      }
   sc.send([c,'',r])
})
```

En este caso, la modificación más significativa consiste en advertir que los mensajes tendrán ahora un segmento más. En nuestro ejemplo del apartado anterior, ese segmento adicional se añadió al final del mensaje. Por tanto, aquí también tendrá que aparecer al final de la lista de parámetros del *callback* que gestiona los eventos 'message'.

En el componente "worker" ya hemos comprobado si la capacidad inicial era positiva o no. Si no lo fuera, el "worker" ni siquiera enviaba un mensaje de registro. Por tanto, en el "broker" sabemos que los mensajes de registro (caso c=='') siempre conllevarán la inclusión del nuevo "worker" como disponible. Para el resto de casos, comprobaremos si la capacidad es todavía positiva o no. Si lo es, seguimos haciendo lo mismo que en el programa original. Si no lo es, simplemente reenviamos la respuesta al cliente correspondiente. Con ello ya gestionamos correctamente todos los casos a considerar. Obsérvese que un "worker" sin capacidad restante no tiene sentido que compruebe si hay clientes esperando, pues él ya no podrá atender esas peticiones encoladas. Tampoco tendrá sentido que lo añadamos a la cola de "workers" disponibles.