

TSR

Examen de recuperación de la Práctica 2 (30 Enero 2025)

Esta prueba se compone de dos preguntas. Requiere obtener el mínimo indicado en la guía docente (3 sobre 10), y contribuye con 3 puntos a la nota final.

1. (5 puntos) (Contesta en papel separado) Dado el código del **publicador** de la práctica 2:

```
1: const {zmq, error, lineaOrdenes, traza, adios, creaPuntoConexion} =
  require('../tsr')
2: lineaOrdenes("port tema1 tema2 tema3")
3: let temas = [tema1,tema2,tema3]
4: let pub = zmq.socket('pub')
5: creaPuntoConexion(pub, port)
6:
7: function envia(tema, numMensaje, ronda) {
8:   traza('envia','tema numMensaje ronda',[tema, numMensaje, ronda])
9:   pub.send([tema, numMensaje, ronda])
10: }
11: function publica(i) {
12:   return () => {
13:     envia(temas[i%3], i, Math.trunc(i/3))
14:     if (i==10) adios([pub],"No me queda nada que publicar. Adios")()
15:     else setTimeout(publica(i+1),1000)
16:   }
17: }
18: setTimeout(publica(0), 1000)
19: pub.on('error', (msg) => {error(`${msg}`)})
20: process.on('SIGINT', adios([pub],"abortado con CTRL-C"))
```

Modifique este programa de manera que respete todas estas condiciones simultáneamente:

- a) Emitirá un mensaje **periódicamente**, alternando cíclicamente entre todos los temas especificados en los argumentos recibidos, sin fin. **(30%)**
- b) El número de temas a utilizar lo decidirá el usuario en cada ejecución, facilitando para ello los argumentos necesarios en la línea de órdenes. **(30%)**
- c) Los mensajes se difundirán cada medio segundo. **(10%)**
- d) No debe utilizarse `setTimeout` y tampoco una variable global para dar el valor de `numMensajes` cuando se invoque la función `envia`. **(30%)**

2. (5 puntos) (Contesta en la página siguiente) Dado el código del broker tolerante a fallos utilizado en la última sesión de la práctica 2:

```
1: const {zmq,lineaOrdenes,traza,error,adios,creaPuntoConexion} = require('../tsr')
2: const ans_interval = 2000
3: lineaOrdenes("frontendPort backendPort")
4: let failed = {}
5: let working = {}
6: let ready = []
7: let pending = []
8: let frontend = zmq.socket('router')
9: let backend = zmq.socket('router')
10: function dispatch(client, message) {
11:   traza('dispatch','client message',[client,message])
12:   if (ready.length) new_task(ready.shift(), client, message)
13:   else pending.push([client,message])
14: }
15: function new_task(worker, client, msg) {
16:   traza('new_task','client message',[client,msg])
17:   working[worker]=setTimeout(()=>{failure(worker,client,msg)}, ans_interval)
18:   backend.send([worker,'', client,'', msg])
19: }
20: function failure(worker, client, message) {
21:   traza('failure','client message',[client,message])
22:   failed[worker] = true
23:   dispatch(client, message)
24: }
25: function frontend_message(client, sep, message) {
26:   traza('frontend_message','client sep message',[client,sep,message])
27:   dispatch(client, message)
28: }
29: function backend_message(worker, sep1, client, sep2, message) {
30:   traza('backend_message','worker sep1 client sep2 message',
31:     [worker,sep1,client,sep2,message])
32:   if (failed[worker]) return
33:   if (worker in working) {
34:     clearTimeout(working[worker])
35:     delete(working[worker])
36:   }
37:   if (pending.length) new_task(worker, ...pending.shift())
38:   else ready.push(worker)
39:   if (client) frontend.send([client,'',message])
40: }
41: frontend.on('message', frontend_message)
42: backend.on('message', backend_message)
43: frontend.on('error' , (msg) => {error(`${msg}`)})
44: backend.on('error' , (msg) => {error(`${msg}`)})
45: process.on('SIGINT' , adios([frontend, backend],"abortado con CTRL-C"))
46: creaPuntoConexion(frontend, frontendPort)
47: creaPuntoConexion(backend, backendPort)
```

(Las cuestiones están en la página siguiente)

Cierto programador ha analizado el código de ese broker y ha sugerido que permite gestionar adecuadamente los siguientes escenarios:

- Reenvío de una petición hacia el primer trabajador disponible, pues hay alguno.
- Encolado de una petición si no hay trabajadores disponibles.
- Reenvío de una respuesta hacia su cliente.
- Reenvío de una petición hacia otro trabajador, cuando falla el trabajador inicialmente asignado.
- Encolado de una petición tras haber fallado el trabajador hacia el que fue reenviada inicialmente, si no hay otros trabajadores disponibles.
- Descarte de una respuesta tardía enviada por un trabajador excesivamente lento.
- Admisión de un mensaje inicial de registro enviado por un nuevo trabajador.
- Llegada, dentro del plazo previsto, de una respuesta emitida por un trabajador.
- Reenvío de una petición encolada hacia un trabajador que acaba de quedar libre.

Identifique (marcando en la tabla) qué escenario o escenarios, de entre los que acabamos de listar, podría/n ocasionar que las condiciones utilizadas en las siguientes líneas llegaran a cumplirse y se ejecutaran sus instrucciones asociadas:

- Línea 12: `if (ready.length) new_task(ready.shift(), client, message)`
- Línea 32: `if (failed[worker]) return`
- Línea 33: `if (worker in working) { ... }`
- Línea 37: `if (pending.length) new_task(worker, ...pending.shift())`
- Línea 39: `if (client) frontend.send([client, '', message])`

(contesta en esta misma tabla con SÍ o NO en cada celda)

	a	b	c	d	e	f	g	h	i
i									
ii									
iii									
iv									
v									