TSR

Examen de la sessió 3 de la Pràctica 2 (13 de gener de 2025)

Donat el codi del **broker tolerant a fallades** utilitzat en la darrera sessió de la pràctica 2:

```
 1: const {zmq,lineaOrdenes,traza,error,adios,creaPuntoConexion} = require('../tsr')
 2: const ans_interval = 2000 // deadline to detect worker failure
 3: lineaOrdenes("frontendPort backendPort")
 4: let failed   = {}  // Map(worker:bool) failed workers has an entry
 5: let working  = {}          // Map(worker:timeout) for workers executing tasks
 6: let ready    = [] // List(worker) ready workers (for load-balance)
 7: let pending  = []          // List([client,message]) requests waiting for workers
 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  // ignore messages from failed nodes
33:         if (worker in working) { // task response in-time
34:                 clearTimeout(working[worker]) // cancel timeout
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)
```

Es vol **afegir un quart component** al sistema CBW tolerant a fallades: l'**operador**. Aquest component s'encarrega de "reparar" els treballadors que el broker detecte que hagen fallat. Per això…

- Quan el broker detecte que un treballador ha fallat, enviarà un missatge a l'operador (a més de la seua reacció normal) amb l'identificador del treballador. Tant el broker com l'operador hauran rebut inicialment des de la línia d'ordres la informació necessària per tal de crear el socket o sockets pertinents per a permetre que aquesta comunicació siga possible. **(30%)**

- La "reparació" d'un treballador per part de l'operador serà una activitat EMULADA mitjançant una espera de 40 segons. Quan l'operador finalitze la reparació d'un treballador, aquest treballador serà reiniciat per l'operador, amb el mateix identificador i de manera automatitzada utilitzant la funció *reiniciar(idWorker)*, que ja se suposa implementada i directament accessible sense necessitat d'importar cap mòdul. **(20%)**

- En ser reiniciat, el treballador es reconnectarà al broker, per la qual cosa l'adaptació del broker ha d'admetre aquesta reconnexió (és a dir, acceptar la recepció d'un missatge de registre per a un treballador que havia fallat prèviament). **(15%)**

- El broker ha de continuar rebutjant una resposta d'un treballador lent si se l'havia donat per caigut i encara no s'ha reconnectat després de la reparació. **(15%)**

- L'operador mostrarà per pantalla la relació de treballadors en reparació (hi pot haver diversos en aquesta situació) cada 5 segons. Observeu que els treballadors ja reparats no han d'aparèixer en aquesta relació. **(20%)**

Se sol·licita **estendre el codi del broker tolerant a fallades, així com desenvolupar el programa operador.js** per implantar el sistema descrit. No caldrà aplicar cap canvi al programa workerReq.js ni al programa utilitzat pels clients.

Les modificacions a aplicar al broker es poden especificar indicant entre quines línies caldria inserir quin codi i quines altres línies caldria eliminar o substituir.

Recordeu que el mòdul `tsr.js` també ofereix una operació `conecta(socket,ip,port)` que podria ser necessària en les extensions del broker o en el codi del programa operador.js.