NIST

Lab 2 Retake Exam (January 30, 2025)

This exam consists of two questions. It requires obtaining the minimum indicated in the teaching guide (3 out of 10), and contributes 3 points to the final grade.

1. (**5 points** . Answer on separate paper) Given the **publisher code** from Lab 2:

```
const {zmq,error,lineaOrdenes,traza,adios,creaPuntoConexion} = require('../tsr')
lineaOrdenes("port tema1 tema2 tema3")
let temas = [tema1,tema2,tema3]
let pub = zmq.socket('pub')
creaPuntoConexion(pub, port)

function envia(tema, numMensaje, ronda) {
  traza('envia','tema numMensaje ronda',[tema, numMensaje, ronda])
  pub.send([tema, numMensaje, ronda])
}
function publica(i) {
  return () => {
    envia(temas[i%3], i, Math.trunc(i/3))
    if (i==10) adios([pub],"No me queda nada que publicar. Adios")()
    else setTimeout(publica(i+1),1000)
  }
}
setTimeout(publica(0), 1000)
pub.on('error', (msg) => {error(`${msg}`)})
process.on('SIGINT', adios([pub],"abortado con CTRL-C"))
```

Modify this program so that it meets all of these conditions simultaneously:

a) It will emit a message **periodically**, cyclically alternating between all the topics specified in the received arguments, without end. (**30%**)

b) The number of topics to be used will be decided by the user in each execution, providing the necessary arguments in the command line. (**30%**)

c) Messages will be broadcast every half second. (**10%**)

d) `setTimeout` should not be used, nor should a global variable be used to give the value of `numMessages when the send` function is invoked. (**30%**)

2 (**5 points**) Answer on the next page) Given the program of the fault-tolerant system broker used in the last session of lab 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)
```

(The questions are on the following page)

STUDENT

LAB GROUP

A certain programmer has analyzed the code of this broker and has suggested that it allows the following scenarios to be properly managed :

a) Forwarding a request to the first available worker, since there is any.
b) Queuing a request if no workers are available.
c) Forwarding a response to a client.
d) Forwarding a request to another worker when the initially assigned worker fails.
e) Queuing a request after the worker to which it was initially forwarded has failed, if no other workers are available.
f) Discarding a late response sent by an excessively slow worker.
g) Acceptance of an initial registration message sent by a new worker.
h) Arrival, within the expected time frame, of a response issued by a worker.
i) Forwarding a queued request to a newly available worker.

Identify (by marking in the table) which scenario(s) from those just listed could cause the conditions used in the following lines to be met and their associated instructions to be executed:

i. Line 12: `if (ready.length) new_task(ready.shift(), client, message)`
ii. Line 32: `if (failed[worker]) return`
iii. Line 33: `if (worker in working) {...}`
iv. Line 37: `if (pending.length) new_task(worker, ...pending.shift())`
v. Line 39: `if (client) frontend.send([client,'',message])`

(answer in this same table with YES or NO in each cell)

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

DON'T FORGET DELIVER THIS SHEET