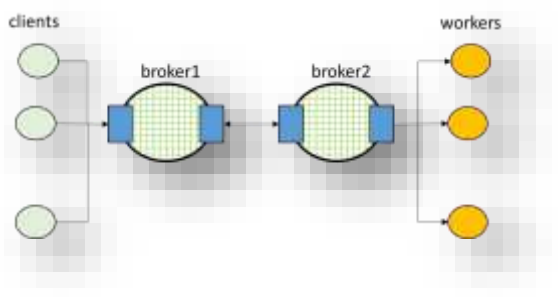
NIST – LAB 2 EXAM, DECEMBER 2, 2021

Question 1 (4 points) These two programs (**broker1** and **broker2**) provide a solution proposal to the division of a broker in two halves, as we have seen in Lab 2.

broker1.js

```
01: const zmq = require('zeromq')
02: let nw=0, cli=[], msg=[]
03: let sc = zmq.socket('router')
04: let sb = zmq.socket(*****)
05: sc.bind('tcp://*:9990')
06: sb.bind('tcp://*:9991')
07:
08: function dispatch(c,m) {
09:    nw--
10:    sb.send([c,'',m])
11: }
12:
13: sc.on('message',(c,sep,m) => {
14:    if (nw!=0) dispatch(c,m)
15:    else {cli.push(c); msg.push(m)}
16: })
17:
18: sb.on('message',(c,sep,r) => {
19:    nw++
20:    if (c!='') sc.send([c,'',r])
21:    //
22: })
```

NOTE.- Let us assume that clients and workers are parameterised with their ID and connection URL. The rest of their programs is identical to that used in the labs.



broker2.js

```
01: const zmq = require('zeromq')
02: let workers=[]
03: let sb = zmq.socket(*****)
04: let sw = zmq.socket('router')
05: sb.connect('tcp://localhost:9991')
06: sw.bind('tcp://*:9992')
07:
08: sw.on('message', (w,sep,c,sep2,r) => {
09:    workers.push(w)
10:    sb.send([c,'',r])
11:
12: })
13:
14: sb.on('message', (c,sep,m) => {
15:   sw.send([workers.shift(),'',c,'',m])
16: })
```

Please, provide a justified answer to these questions:

a) Choose the socket type to be used in broker1's `sb` and broker2's `sb` sockets (declared in line 4 of broker1 and line 3 of broker2). Explain your answer.
b) Justify whether these provided programs work correctly in those situations related with workers availability.
c) Justify whether additional code is needed at line 21 of broker 1 in order to implement a correct solution to the "two halves" broker problem. If so, write that code.

Question 2 (1 point) The **fault-tolerant broker** version used in Lab 2 may handle several error scenarios. Answer these questions related to that system:

a) Explain what happens when the broker fails. Describe its effects on the other components, requests being processed and the complete system.
b) Explain what happens when a worker fails. Describe the differences in these two failure scenarios: i) the worker is processing a request, ii) the worker is waiting for requests.
c) Explain what happens if a worker, assumed faulty by the broker, sends back a late response r to a request m from a client c.

Question 3 (4 points) Let us consider a **chat system** identical to that described in part 5 of Lab 2, with this source code for its client program:

```
01: const zmq = require('zeromq')
02: const nick='Ana' //Assume it's random.
03: let sub = zmq.socket('sub')
04: let psh = zmq.socket('push')
05: sub.connect('tcp://127.0.0.1:9998')
06: psh.connect('tcp://127.0.0.1:9999')
07: sub.subscribe('')
08: sub.on('message', (nick,m) => {
09:    console.log('['+nick+']'+m)
10: })
11: process.stdin.resume()
12: process.stdin.setEncoding('utf8')
```

```
13: process.stdin.on('data'  ,(str) => {
14:    psh.send([nick, str.slice(0,-1)])
15: })
16: process.stdin.on('end',() => {
17:    psh.send([nick, 'BYE'])
18:    sub.close(); psh.close()
19: })
20: process.on('SIGINT',() => {
21:    process.stdin.end()
22: })
23: psh.send([nick,'HI'])
```

One of the participants in that chat (the "*mafioso*") has designed a way for abusing of that system with fake chat clients (the "*minions*") that obey its commands, as follows...

- When mafioso reads a chat message (*original_message*) written by a concrete client (let us call it, *target*), mafioso reacts commanding each minion to send a chat message with this content: "**I don't like the message from *target*: *original_message*".**
- Both *mafioso* and *minion* should be new versions of the generic chat client. **Their implementation only adds other lines to the original code**, with a clean separation between the original lines and the extensions needed in the *mafioso-minions* communication.
    - With a single exception: the lines that handle the *mafioso* reaction.
- *Mafioso* and *minion* receive from the command line, in their variable ***port***, the port number to be used in their intercommunication. *Mafioso* also receives in its variable ***target*** the identifier of the user to be disturbed. You do not need to write any instruction to assign those initial values to those variables in the subsequent questions c) and d).

Please answer these questions about *mafioso* and *minion*:

a) Choose the ZeroMQ socket type(s) to communicate *mafioso* and *minions*. Explain your answer.

b) The *mafioso* program is based on the chat client. Which are the instructions to be inserted in that program in order to detect a message from *target* and start the *mafioso* reaction? State where they should be written. Use line numbers to this end.

c) In *mafioso*, write the instructions that extend the base chat client in order to provide the remaining functionality of *mafioso*.

d) In *minion*, write the instructions that extend the base chat client in order to provide the *minion* functionality.

Question 4 (1 point) A part of Lab 2 requires a periodical visualisation of **statistics on the handled requests**, but in this question their total number should not be considered. Explain which data structures are needed for handling **per-worker** data, and how those data are accessed. Write the function that, according to the bulletin, shows that information every 5 seconds (e.g., function `visualize()`, started with `setInterval(visualize, 5000)`)