PART 1 (6 PUNTS)

A continuació es mostren els programes *mybroker.js, myclient.js* i *myworker.js* que faciliten la base inicial per a desenvolupar la segona part de la segona pràctica. Són idèntics als presentats en el butlletí.

Prenent-los com a base, resol les 3 activitats proposades al final.

```
1:  // ROUTER-ROUTER request-reply broker in NodeJS
2:  var zmq      = require('zmq')
3:    , frontend = zmq.socket('router')
4:    , backend  = zmq.socket('router');
5:
6:  var fePortNbr = 8059;
7:  var bePortNbr = 8060;
8:  var workers = [];
9:  var clients = [];
10:
11:   frontend.bindSync('tcp://*:'+fePortNbr);
12:   backend.bindSync('tcp://*:'+bePortNbr);
13:
14:   frontend.on('message', function() {
15:     var args = Array.apply(null, arguments);
16:     if (workers.length > 0) {
17:       var myWorker = workers.shift();
18:       var m = [myWorker,''].concat(args);
19:       backend.send(m);
20:     } else
21:       clients.push( {id: args[0],msg: args.slice(2)});
22:   });
23:
24:   function processPendingClient(workerID) {
25:     if (clients.length>0) {
26:       var nextClient = clients.shift();
27:       var m = [workerID,'',nextclient.id,''].concat(nextClient.msg);
28:       backend.send(m);
29:       return true;
30:     } else
31:       return false;
32:   }
33:
34:   backend.on('message', function() {
35:     var args = Array.apply(null, arguments);
36:     if (args.length == 3) {
37:       if (!processPendingClient(args[0]))
38:       workers.push(args[0]);
39:     } else {
40:       var workerID = args[0];
41:       args = args.slice(2);
42:       frontend.send(args);
43:       if (!processPendingClient(workerID))
44:         workers.push(workerID);
45:     }
46:   });
```

```
1:  // myclient in NodeJS (myclient.js)
2:  var zmq       = require('zmq')
3:    , requester = zmq.socket('req');
4:
5:  var brokerURL = 'tcp://localhost:8059';
6:  var myID = 'NONE';
7:  var myMsg = 'Hello';
8:
9:  if (myID != 'NONE')
10:     requester.identity = myID;
11:   requester.connect(brokerURL);
12:   console.log('Client (%s) connected to %s', myID, brokerURL)
13:
14:   requester.on('message', function(msg) {
15:     console.log('Client (%s) has received reply "%s"', myID, msg.toString());
16:     process.exit(0);
```

```
17:   });
18:   requester.send(myMsg);
```

```
1:  // myworker server in NodeJS (myworker.js)
2:  var zmq = require('zmq')
3:    , responder = zmq.socket('req');
4:
5:  var backendURL = 'tcp://localhost:8060';
6:  var myID = 'NONE';
7:  var connText = 'id';
8:  var replyText = 'World';
9:
10:   if (myID != 'NONE')
11:     responder.identity = myID;
12:   responder.connect(backendURL);
13:   responder.on('message', function(client, delimiter, msg) {
14:     setTimeout(function() {
15:       responder.send([client,'',replyText]);
16:     }, 1000);
17:   });
18:   responder.send(connText);
```

Es pretén modificar aquesta solució perquè el sistema es comporte d'aquesta manera:

Quan un client envie una petició i no hi haja treballadors disponibles en la cua **workers[]**, no esperarà en cap cua. En el seu lloc, rep una resposta especial: **'NoWorkers'**. En rebre aquesta resposta, el client programa un reintent en un segon i arribarà a reintentar fins a cinc vegades; si aquesta darrera resposta encara és **'NoWorkers'** mostra un missatge d'error a l'usuari ("**All workers are busy at the moment!**") i acaba.

Les dues primeres activitats requereixen modificacions en els programes. Has d'especificar quins canvis es necessiten per a implantar el que es propose. Per a fer això:

- Especifica clarament quines línies dels programes originals han de modificar-se.
- En cas d'afegir més codi, indica entre quines dues línies ha de ser inserit.
- Alternativament, pots escriure el programa complet.

Es demana realitzar aquestes 3 activitats:

1. (2 punts) Escriu les modificacions necessàries en *myclient.js*.
2. (2 punts) Escriu les modificacions necessàries en *mybroker.js*.
3. (2 punts) Suposem que en una terminal Linux s'han escrit les següents línies d'ordres, sense cap pausa entre elles. Explica quants clients mostraran el missatge d'error ("**All workers are busy at the moment!**"):

```
$ node mybroker&
$ node myworker&
$ node myclient& node myclient& node myclient& node myclient& node myclient&
node myclient& node myclient& node myclient&
```

(les 2 últimes línies es refereixen a una única ordre per a llançar 8 clients simultàniament)

SOLUCIÓ PART 1

1. El component myclient.js necessita una variable global (per exemple, attempts) inicialitzada a zero per a conèixer el nombre d'intents que s'han realitzat fins al moment. Això es pot fer inserint una nova línia, entre les originals 7 i 8, amb aquest contingut:

```
var attempts = 0;
```

Addicionalment, el cos del *listener* per als esdeveniments "message" (línies 15 i 16) ha de ser reemplaçat amb les línies 20 a 26 de la solució completa:

```
01:  // myclient in NodeJS (myclient.js)
02:  var zmq       = require('zmq')
03:    , requester = zmq.socket('req');
04:
05:  var brokerURL = 'tcp://localhost:8059';
06:  var myID = 'NONE';
07:  var myMsg = 'Hello';
08:  var attempts = 0;
09:
10:  if (myID != 'NONE')
11:    requester.identity = myID;
12:  requester.connect(brokerURL);
13:  console.log('Client (%s) connected to %s', myID, brokerURL)
14:
15:  requester.on('message', function(msg) {
16:    if (msg != 'NoWorkers') {
17:      console.log('Client (%s) has received reply "%s"', myID, msg.toString());
18:      process.exit(0);
19:    }
20:    else {
21:      attempts++;
22:      if (attempts==6) {
23:        console.log("All workers are busy at the moment!");
24:        process.exit(1);
25:      }
26:      setTimeout( function() { requester.send(myMsg) }, 1000);
27:    });
28:  requester.send(myMsg);
```

2. El nou broker no necessita la cua de clients, ja que en lloc d'esperar quan no hi ha workers disponibles els clients ara reben una resposta especial del broker. Això significa que en la nova versió de `mybroker.js` les línies originals 9, 24-33, 37 i 43 poden eliminar-se opcionalment. No obstant això, la següent substitució de la línia 21 és necessària:

```
frontend.send([args[0],'','NoWorkers'])
```

   - Això assegura que cap client estarà en la cua "clients" i que rebran la resposta "NoWorkers" quan siga necessari.

Així, la nova versió d'aquest broker és:

```
01:  // ROUTER-ROUTER request-reply broker in NodeJS
02:  var zmq      = require('zmq')
03:    , frontend = zmq.socket('router')
04:    , backend  = zmq.socket('router');
05:
06:  var fePortNbr = 8059;
07:  var bePortNbr = 8060;
08:  var workers = [];
09:
10:  frontend.bindSync('tcp://*:'+fePortNbr);
11:  backend.bindSync('tcp://*:'+bePortNbr);
12:
13:  frontend.on('message', function() {
14:    var args = Array.apply(null, arguments);
15:    if (workers.length > 0) {
16:      var myWorker = workers.shift();
```

```
17:       var m = [myWorker,''].concat(args);
18:       backend.send(m);
19:     } else
20:       frontend.send([args[0],'','NoWorkers']);
21:   });
22:
23:   backend.on('message', function() {
24:     var args = Array.apply(null, arguments);
25:     if (args.length == 3) {
26:         workers.push(args[0]);
27:     } else {
28:         var workerID = args[0];
29:         args = args.slice(2);
30:         frontend.send(args);
31:         workers.push(workerID);
32:     }
33:   });
```

3. Analitzem el que cada procés està fent en cada instant en l'execució de tots els components:

Seg.	Worker	Broker	Client 1	Client 2	Client 3	Client 4	Client 5	Client 6	Client 7	Client 8
0	Cli1 req rcvd.	Cli1 req. to worker	Req. sent	Req. sent / 'NoWrk' rcvd.	Req. sent / 'NoWrk' rcvd.	Req. sent / 'NoWrk' rcvd.	Req. sent / 'NoWrk' rcvd.	Req. sent / 'NoWrk' rcvd.	Req. sent / 'NoWrk' rcvd.	Req. sent / 'NoWrk' rcvd.
1	Cli1 replied / Cli2 rcvd.	Cli2 req. to worker	Reply rcvd.	Req. resent	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.
2	Cli2 replied / Cli3 rcvd.	Cli3 req. to worker		Reply rcvd.	Req. resent	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.
3	Cli3 replied / Cli4 rcvd.	Cli4 req. to worker			Reply rcvd.	Req. resent	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.
4	Cli4 replied / Cli5 rcvd.	Cli5 req. to worker				Reply rcvd.	Req. resent	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.
5	Cli5 replied / Cli6 rcvd.	Cli6 req. to worker					Reply rcvd.	Req. resent	Req. resent / 'NoWrk' rcvd.	Req. resent / 'NoWrk' rcvd.
6	Cli6 replied	Cli6 reply forw.						Reply rcvd.	Error shown to user	Error shown to user

Com es pot observar en la taula, quan l'execució comença, un dels vuit clients aconsegueix entregar primer la seua petició al broker. Anomenarem a aquest client "Client 1". La resta de peticions reben "un NoWorkers" com a resposta. Això provoca que tots aquests clients reprogramen l'enviament de la seua petició. Considerem que en aquest mateix instant es produeix la resposta del worker a la petició de "Client 1". Aquest rep la resposta i els altres set estan enviant la seua petició. Un d'ells trobarà al worker en estat disponible. Els sis restants obtindran la resposta "NoWorkers". Han usat el primer dels seus cinc reintents.
En l'instant 2, el client 2 rep la seua resposta i els 6 restants envien de nou la seua petició. Un d'ells serà servit. Això significa que en cada instant se serveix un client. Atès que cada client envia la seua petició original més cinc reintents, cap d'ells mostra errors fins a l'instant 6. En aquest instant 6 clients han obtingut la seua resposta i només es mostraran dos errors.

PART 2 (2 PUNTS: 0.5 PUNTS PER QÜESTIÓ)

Suposem un sistema en el qual s'utilitzen les versions originals dels programes mostrats en la Part 1 (és a dir, *myclient.js, mybroker.js* i *myworker.js*). Explica què succeeix quan els processos que s'esmenten en les qüestions següents són iniciats i les seues respectives execucions acaben o romanguen suspeses (a causa que no hi ha nous esdeveniments que processar).

Per a fer això, indica si cada procés ha...:

- **Clients**: (**a**) sigut incapaç de comunicar-se amb altres processos, (**b**) deixat el seu missatge en una cua del broker, (**c**) obtingut una resposta, o (**d**) acabat la seua execució de manera abrupta.
- **Treballadors**: (**a**) sigut incapaç de comunicar-se amb altres processos, (b) deixat la seua identitat en una cua del broker, (**c**) transmès satisfactòriament la seua resposta al client adequat, o (**d**) acabat la seua execució de manera abrupta.
- **Brokers**: (**a**) sigut incapaç de gestionar els missatges enviats pels clients, (**b**) sigut incapaç de gestionar els missatges enviats pels treballadors, (**c**) funcionat sense cap problema fins al moment, o (**d**) acabat la seua execució de manera abrupta.

Els casos a considerar són (cadascun assumeix que no hi havia processos en marxa):

1. **S'inicien 3 clients. Un segon després, s'inicia el bróker. No es llança cap treballador.**

   Clients (b): Tots els clients han inserit en la cua "clients" la seua petició. Estan esperant la seua resposta que arribarà si en un futur pròxim es posen workers en funcionament. La connexió amb el broker tardarà cert temps, però finalment es produirà sense cap error ni excepció.
   Broker (c): Ha rebut tres peticions dels clients (una per cada client) i les ha emmagatzemades en el vector "clients". Atès que no hi ha cap worker, les peticions romanen en el vector. El broker funciona correctament i enviarà les peticions quan es pose un worker en execució.

2. **S'inicien 2 treballadors. Després de 3 segons, s'inicien 3 clients. No comença cap bróker.**

   Clients (a): Els clients han intentat connectar-se al broker i han enviat les seues peticions. Les peticions seran processades quan s'inicie un broker. De moment, cap client ha pogut comunicar amb cap component.
   Treballadors (a): Els workers han intentat connectar amb el broker i han enviat els seus missatges de registre inicial. Aquests intents seran processats quan es pose en execució un broker. De moment, cap worker s'ha pogut comunicar amb cap component.

3. **S'inicia un bróker A. Un segon després, s'inicia un bróker B. De moment no es llança cap client ni treballador.**

   Broker A (c): El primer broker s'ha iniciat i executa la instrucció BIND correctament en els seus dos sockets ZMQ de tipus ROUTER. Es troba esperant missatges als sockets. El seu comportament és correcte fins al moment.
   Broker B (d): El segon broker generarà una excepció quan intente executar la instrucció bind sobre el port prèviament usat per l'anterior broker. Com aquesta excepció no és tractada en el codi, aquest component acabarà la seua execució.

4. **S'inicia un bróker. Un segon després, 2 treballadors A i B són iniciats, en aquest ordre. 2 segons més tard, es llança un client.**

   Broker (c): Funciona correctament.
   Treballadors (b): Tots dos s'han registrat en el broker. El treballador A ha respost a la petició del client. Després, el seu ID s'emmagatzema en el vector "workers". L'ID del treballador B passa al vector "workers" i no rep cap petició de cap client. Per tant, l'ID de tots dos workers roman en el vector "workers".
   Client (c): El client rep una resposta a la seua petició per part del treballador A.

PART 3 (2 PUNTS: 0.5 PUNTS PER ENCERT, -0.167 PUNTS PER RESPOSTA ERRÒNIA)

Selecciona l'opció correcta. Hi ha una única opció correcta en cada qüestió.

1. Sobre la proposta de solució per a batec, indica quina de les següents afirmacions és CERTA:

	El bróker envia un missatge a tots els workers, que han de contestar dins d'un termini de temps. El broker no fa cap enviament de missatge a tots els workers. Tan sols reenvia cada petició rebuda a un worker.
X	Els workers amb treball han de contestar amb el resultat dins d'un termini de temps. El broker estableix un 'timeout' per a cada petició reenviada. Si la resposta corresponent no es rep en l'interval marcat, el broker considera que el worker ha fallat i reenvía la petició a qualsevol dels brokers disponibles. Si no hi ha workers disponibles, aquesta reexpedició es guarda en el vector "clients".
	Els clients reenvien les peticions no finalitzades dins del termini de temps. Els clients no reenvien res, és el broker qui controla un termini de temps de resposta per part dels workers.
	Els workers envien periòdicament missatges al bróker per a confirmar que es troben en funcionament. El worker no envia missatges periòdics.

2. Sobre la proposta de solució per a classes de treball, indica quina de les següents afirmacions és FALSA:

	Excepte el bróker, tots els altres components es limiten a un únic tipus de treball. Cert. En la versió original, tant clients com workers, una vegada iniciats només són capaços de gestionar un únic tipus de treball.
	En el primer missatge enviat per un worker al bróker, el primer informa sobre el tipus de treball que pot atendre. Cert. Els treballadors s'identifiquen amb el seu tipus de treball al broker en el primer missatge que envien.
X	En el primer missatge enviat pel bróker a un worker, el primer informa sobre el tipus de treball que ha d'atendre. Fals. El broker només envia peticions dels clients als workers. No s'envia cap missatge especial, com el que s'indica en aquest apartat.
	Modificant únicament el codi del client, aquest podria canviar el tipus de treball per a cada missatge sense provocar que el sistema falle. Cert. Aquesta modificació és possible. Atès que el client inclou el tipus de treball en un segment del seu missatge (en l'últim de cada petició), si s'estén el codi del client es pot especificar un tipus de treball diferent per a cada petició.

3. El bróker gestiona dues cues relacionades amb clients i treballadors (workers). Selecciona l'afirmació FALSA:

X	La cua de clients pot disposar de dos elements que únicament es diferencien en el contingut del missatge. Fals. Això implicaria que hi ha dues peticions del mateix client en la cua "clients". Això no pot succeir, atès que un client només envia una petició cada vegada.
	Quan una cua té elements, l'altra es troba buida. Cert. La cua "clients" només s'usa quan no hi ha cap worker disponible (és a dir que no hi ha workers en la cua "workers"). Anàlogament només s'encuen treballadors en la cua "workers" quan no hi ha peticions pendents de servir (la cua "clients" està buida).

	No totes les peticions de clients passen per la seua cua clients[]. Cert. Les peticions dels clients es mantenen en la cua "clients" sempre que no hi haja treballadors disponibles. Si en rebre la petició del client hi ha algun treballador lliure, la petició no passa per la cua "clients".
	No tots els missatges de petició (o oferiment inicial) de servei de treballadors passen per la seua cua workers[]. Cert. Si existeix una petició d'algun client, prèvia al missatge enviat pel treballador, llavors al treballador se li assigna aquesta petició sense inserir el seu ID en la cua "workers".

4. Indica si en el nostre sistema client-broker-worker el worker pot enviar missatges amb diferent <u>quantitat de segments:</u>

	No, perquè el codi del client no està preparat per a aquest cas. Fals. El codi del client pot gestionar les respostes quan contenen diferents quantitats de segments. Els clients només estan interessats en un segment del missatge. No obstant això, no produeixen cap excepció quan el missatge està buit o té més segments.
	No, perquè el codi del bróker no està preparat per a aquest cas. Fals. El broker està preparat per a processar missatges de diferents tipus del worker: el seu missatge inicial de disponibilitat i les respostes als clients. Tots dos tenen un nombre diferent de segments.
X	Sí, perquè a voltes el missatge del worker no conté la resposta a una sol·licitud. Cert. El missatge inicial del worker té tres segments, mentre que les respostes als clients tenen més de tres segments.
	Sí, perquè depèn del nombre de segments que tinga la petició que arribe al worker. Fals. Els treballadors només processen un dels segments en la petició dels clients. Per tant, la resposta dels treballadors no depèn de la quantitat de segments de la petició del client.