TSR/NIST- EXAM OF 2ND LAB PROJECT. DECEMBER 11, 2017

The following boxes show the *mybroker.js, myclient.js* and *myworker.js* programs that provide the initial base for developing the second part of the second lab project. They are identical to those presented in the bulletin.

Taking those programs as a base, please provide a solution for each of the three activities proposed afterwards:

```
// ROUTER-ROUTER request-reply broker in NodeJS
var zmq      = require('zmq')
  , frontend = zmq.socket('router')
  , backend  = zmq.socket('router');

var fePortNbr = 8059;
var bePortNbr = 8060;
var workers = [];
var clients = [];

frontend.bindSync('tcp://*:'+fePortNbr);
backend.bindSync('tcp://*:'+bePortNbr);

frontend.on('message', function() {
  var args = Array.apply(null, arguments);
  if (workers.length > 0) {
    var myWorker = workers.shift();
    var m = [myWorker,''].concat(args);
    backend.send(m);
  } else
    clients.push( {id: args[0],msg: args.slice(2)});
});

function processPendingClient(workerID) {
  if (clients.length>0) {
    var nextClient = clients.shift();
    var m = [workerID,'',nextClient.id,''].concat(nextClient.msg);
    backend.send(m);
    return true;
  } else
    return false;
}

backend.on('message', function() {
  var args = Array.apply(null, arguments);
  if (args.length == 3) {
    if (!processPendingClient(args[0]))
      workers.push(args[0]);
  } else {
      var workerID = args[0];
      args = args.slice(2);
      frontend.send(args);
      if (!processPendingClient(workerID))
        workers.push(workerID);
  }
});
```

```
// myclient in NodeJS (myclient.js)
var zmq       = require('zmq')
  , requester = zmq.socket('req');

var brokerURL = 'tcp://localhost:8059';
var myID = 'NONE';
var myMsg = 'Hello';

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);
```

```
01:   // myworker server in NodeJS (myworker.js)
02:   var zmq = require('zmq')
03:     , responder = zmq.socket('req');
04:
05:   var backendURL = 'tcp://localhost:8060';
06:   var myID = 'NONE';
07:   var connText = 'id';
08:   var replyText = 'World';
09:
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);
```

Let us modify this solution in order to achieve the following behaviour:

When a client sends a request and there are no available workers in the **workers[]** queue, it doesn't remain awaiting in a queue. Instead, it receives a special answer: **'NoWorkers'**. When such a reply is received, the client programs a retry of that request in a second. The client retries up to 5 times; if that fifth attempt still returns **'NoWorkers'** it reports an error message to the user ("**All workers are busy at the moment!**") and ends.

The first two activities require several program modifications. In those cases, you must specify which changes are needed for implementing what is being proposed. To this end:

- State clearly which lines of the original programs need to be changed, if any.
- In case of inserting new code, indicate between which lines it needs to be inserted.
- Alternatively, you may write a complete new program.

The three activities in this Part 1 are:

1. (2 points) Write the modifications being needed in program *myclient.js*.
2. (2 points) Write the modifications being needed in program *mybroker.js*.
3. (2 points) Let us assume a Linux terminal where the following commands have been run in the shown order, without any delay between consecutive lines. Discuss how many client processes will show the error message mentioned above ("**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 &
```

(last 2 lines must be interpreted as a single command to run 8 clients simultaneously)

1. The new client program requires a new global variable (e.g., “attempts”) initialised to zero in order to know how many attempts have been done up to now. This might be done inserting a new line with these contents: “var attempts=0;” between the original lines 7 and 8. Additionally, the body of the listener for “message” events (originally placed in the lines 15 to 16) should be replaced with these lines:

```
  if (msg!='NoWorkers') {
        console.log('Client (%s) has received reply "%s"', myID, msg.toString());
        process.exit(0);
  } else {
        attempts++;
        if (attempts==6) {
                console.log("All workers are busy at the moment!");
                process.exit(1);
        }
        setTimeout( function() { requester.send(myMsg) }, 1000);
  }
```

Thus, the resulting complete program is:

```
// myclient in NodeJS
var zmq = require('zmq')
, requester = zmq.socket('req');
var brokerURL = 'tcp://localhost:8059';
var myID = 'NONE';
var myMsg = 'Hello';
var attempts = 0;

if (myID != 'NONE')
        requester.identity = myID;
requester.connect(brokerURL);
console.log('Client (%s) connected to %s', myID, brokerURL)

requester.on('message', function(msg) {
  if (msg!='NoWorkers') {
        console.log('Client (%s) has received reply "%s"', myID, msg.toString());
        process.exit(0);
  } else {
        attempts++;
        if (attempts==6) {
                console.log("All workers are busy at the moment!");
                process.exit(1);
        }
        setTimeout( function() { requester.send(myMsg) }, 1000);
  }
});

requester.send(myMsg);
```

2. The new broker program will not need any “clients” queue, since now clients receive a special reply from the broker when there are no available workers. This means that the original lines 9, 24-33, 37 and 43 may be suppressed in the new version of the mybroker.js program. Besides, line 21 must be replaced with the following contents: “frontend.send([args[0],’’,’NoWorkers’])”. This ensures that no client will ever be in the “clients” queue and that they receive the ‘NoWorkers’ reply when needed. Therefore, that single change in line 21 is the key mandatory update in this program. All other removals may be considered optional.
Thus, a new short version of this program is:

```
// ROUTER-ROUTER request-reply broker in NodeJS.
var zmq = require('zmq')
, frontend = zmq.socket('router')
, backend = zmq.socket('router');

var fePortNbr = 8059;
var bePortNbr = 8060;
var workers = [];

frontend.bindSync('tcp://*:'+fePortNbr);
backend.bindSync('tcp://*:'+bePortNbr);

frontend.on('message', function() {
        var args = Array.apply(null, arguments);
        if (workers.length > 0) {
                var myWorker = workers.shift();
                var m = [myWorker,''].concat(args);
                backend.send(m);
        } else
                frontend.send([args[0],'','NoWorkers']);
        });

backend.on('message', function() {
        var args = Array.apply(null, arguments);
        if (args.length != 3) {
                workers.push(args[0]);
        } else {
                var workerID = args[0];
                args = args.slice(2);
                frontend.send(args);
                workers.push(workerID);
        }
});
```

3. Let us analyse what each process is doing at each second in the execution of all those agents:

Time	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

As we can see in the table, when the execution is starting, one of the eight clients is able to deliver first its request to the broker. Let us call "Client 1" this client. Its request is forwarded to the worker. However, all requests from the remaining clients receive a "NoWorkers" string as a reply. This means that all those clients reprogram the sending of their requests in a second. That second is also the time being used by the worker for sending its reply to Client 1.
Therefore, more or less at the same time, at second 1 in this trace, Client 1 receives its answer and the other seven clients send again their requests. One of those seven clients finds the worker in an available state. Thus, its request is forwarded to that worker. The remaining six clients receive again a 'NoWorkers' reply. All they have used the first of their five possible retries.
At second 2, Client 2 receives its answer and the other six clients send again their requests. One of them is served. This means that this system is able to serve one client per second. Since each client is able to send its original request plus up to five reattempts, then none of them shows any error message to its user until second 6. At that time, six clients have already received their intended answer. As a result of this, only two errors are shown.

NAME:		SURNAME:			
SIGNATURE:		ID# (DNI):		LAB GROUP:	

PART 2 (2 points: 0.5 points per question)

Let us assume a system that uses the original versions of the programs shown in Part 1 (i.e., *myclient.js, mybroker.js* and *myworker.js*). Discuss what happens when the set of processes described in each case is started and their respective executions either end or remain stopped (due to the lack of any other events to be processed), choosing whether they have...:

- **Clients**: (**a**) Been unable to communicate with the other agents, (**b**) placed their message in a broker queue, (**c**) obtained a reply, or (**d**) terminated abnormally.
- **Workers**: (**a**) Been unable to communicate with the other agents, (**b**) placed their identity in a broker queue, (**c**) successfully forwarded their reply to their intended client, or (**d**) terminated abnormally.
- **Brokers**: (**a**) Been unable to manage the messages sent by clients, (**b**) been unable to manage the messages sent by workers, (**c**) terminated abnormally or (**d**) been operating as intended.

The cases to be considered are the following:

1. **Three clients are started. One second later, the broker is started. No worker is started.**

   Clients (b): All they have placed their message in the “clients” queue of the broker. They are waiting for their reply that will eventually arrive if a worker is started in the future. The connection with the broker took some time, but it was eventually set without any error or exception.
   Broker(d): It has received three request messages from clients (one per client) and it has stored those requests in its “clients” array. Since no worker has been started, those requests are kept in that array. The broker has behaved correctly and it would forward those requests when at least one worker started.

2. **Two workers are started. Three seconds later, three clients are started. No broker is started.**

   Clients (a): Clients have tried to connect with the broker and have sent their respective requests. Those connection attempts and sent messages will be eventually processed when a broker starts. At the moment, each client has still been unable to communicate with any other agent.
   Workers (a): The workers have tried to connect with the broker and have sent their respective registration messages. Those connection attempts and sent messages will be eventually processed when a broker starts. At the moment, each worker has still been unable to communicate with any other agent.

3. **One broker A is started. One second later, another broker B is started. No clients or workers are started.**

   Broker A (d): The first broker has been started and has been able to bind their two ROUTER sockets to their corresponding ports. It is waiting for incoming messages. Its behaviour has been correct until now.
   Broker B (c): The second broker has generated an exception when it tried to bind their first socket, since that port was already busy (it is being used by Broker A). Since that exception is not handled, this agent aborts in that instruction.

4. **One broker is started. One second later, two workers A and B are started, in that order. Two seconds later, one client is started.**

   Broker (d): It is working correctly.
   Workers (b): Both have been registered in the broker. Worker A has replied to a client request. Later, its ID has been kept in the workers array. Worker B put its identity in that array and has not received any client request yet. Therefore, both identities are in the workers array, now.
   Client (c): This client has received a reply to its request. That reply was generated by worker A.

PART 3 (2 points: 0.5 points per correct answer, -0.167 points per wrong answer)

Select the correct choice. There is a single correct choice per question.

1. **On the bulletin proposal of a solution for <u>heartbeat</u> management, choose the TRUE sentence:**

	The broker sends a message to all workers, that each worker must reply in a given time interval. The broker doesn't broadcast any message to all workers. It only forwards each incoming request to a single worker.
X	Each worker that is processing any request must reply with its answer in a given time interval. The broker sets a timeout for each forwarded request. If its corresponding reply isn't received in that interval, the broker considers that such a worker has failed and resends that request to any of the available workers. If no worker is available, that reattempt is placed in the clients array.
	Clients resend their requests that haven't been replied in a given time interval. Those requests are resent, but those replays are managed by the broker. Such a management doesn't need any client intervention.
	Each worker sends a periodical message to the broker in order to report that it still works properly. No explicit periodical message needs to be sent by workers.

2. **On the bulletin proposal of a solution for <u>job classes</u>, choose the FALSE sentence:**

	With the exception of the broker, all other agents are limited to a single job class. True. In their original version, both clients and workers –once started— are only able to manage requests from a single job class.
	In the first message sent by a worker to the broker, the former reports which class of job it may manage. True. Workers report their identity and job class to the broker in their first sent message.
X	In the first message sent by the broker to a worker, the former states which class of job the latter must service. False. Brokers only forward client request messages to workers. Brokers do not send any other kind of message to workers.
	With a modification in its code, the client may change the job class for each of its requests and the resulting system will still work without any problem. True. That modification is feasible. Client requests include the job class as one of their message segments (concretely, in the last segment of each request). If the client program is extended, it might specify a different job class in each of its requests. For instance, receiving the list of job classes to be used as additional command-line arguments. No modification is needed in the broker code for managing those requests, since the broker never records which job class has been used up to now by each client.

3. **The broker manages two <u>queues</u> related to clients and workers. Choose the FALSE sentence:**

X	The client queue may have two elements that only differ in the contents of their message. False. This means that two messages from the same client have been kept in the "clients" queue. That cannot happen, since each client only sends a single message

	When one of those queues is not empty, the other is empty. True. The “clients” queue is only used when there is no worker available. This means that the “workers” queue is empty in those cases. On the other hand, the “workers” queue is only used when there are no pending client requests in the “clients” queue.
	Not all client requests have been in the clients[] queue. True. Client requests are only temporarily kept in the “clients” queue if there are no available workers when they are received. If a client request is received when there is at least one worker ID in the “workers” queue, then such client request doesn’t use the “clients” queue.
	Not all worker “requests” (i.e., their initial availability reports) have been in the workers[] queue. True. If there is at least one client request in the “clients” queue, then an incoming worker availability report will not be in the “workers” queue. It is associated to the oldest client request and such a request is immediately forwarded to that available worker.

4. Discuss whether in our client-broker-worker system the worker may send messages with a <u>different amount of segments</u>:

	No, since the client code cannot manage that scenario. False. The client code may manage without any problem replies with different amounts of segments. Clients are only interested in one message segment. In spite of this, they do not throw any exception when the message is empty or has more than one segment.
	No, because the broker code cannot manage that scenario. False. The broker is ready for processing two different types of worker messages: their initial availability reports, and their client replies. Those two types of messages differ in their amount of segments.
X	Yes, since sometimes the worker message doesn’t carry the reply to a client request. True. Those worker initial availability reports have three segments, while client replies have more than three segments.
	Yes, because the amount of segments in a worker message depends on how many segments had the client message received by that worker. False. Workers only consider one of the client request segments. Thus, worker replies do not depend on the amount of segments contained in each client request.