TSR: Activitats del Seminari 3

ACTIVITAT 1

OBJECTIU: Experimentar amb l'ús de bind() i connect()

ENUNCIAT: Prenent com a base l'exemple de la transparència 33:

```
// Client
var zmq = require('zmq');
var rq = zmq.socket('req');

rq.connect('tcp://127.0.0.1:8888');

rq.send('Hello');

rq.on('message', function(msg) {
  console.log('Response: ' + msg);
});
```

```
// Server
var zmq = require('zmq');
var rp = zmq.socket('rep');

rp.bind('tcp://127.0.0.1:8888',function(err) {
  if (err) throw err;
});

rp.on('message', function(msg) {
  console.log('Request: ' + msg);
  rp.send('World');
});
```

1. Modificar el client perquè envie un missatge periòdicament.
2. Realitzar una nova modificació del client. Ara el missatge ha d'incorporar un nombre de seqüència, perquè així es puga distingir entre cada missatge rebut.
3. Executar el client modificat i el servidor en processos diferents.
    a. Modifique el servidor de manera que acabe (sense respondre) després d'haver rebut la desena petició.
    b. Reinicie el servidor.
    c. Descriga què ocorre en el client.

Modifiquem ara tots dos programes com segueix:

- El client farà un bind() del seu socket, en lloc d'un connect().

- El servidor realitzarà un connect(), en lloc d'un bind().

Ara, igual que abans, execute client i servidor. Acabe el servidor. Llance el servidor de nou.

4. Descriga les diferències en el comportament del client, en cas que n'observe alguna.
5. Discutisca quins avantatges i inconvenients aporten aquests canvis en els bind() i connect().
6. Presente el codi resultant.

ACTIVITAT 2

OBJECTIU: Adaptar el patró petició-resposta dels sockets a l'ús de promeses.

ENUNCIAT: En el patró petició-resposta, l'agent que usa el socket **req** és el client, mentre que l'agent amb el socket **rep** és el servidor.

Com ja s'ha presentat en classe, l'API gestiona el missatge de resposta com un esdeveniment ("message") i els problemes potencials amb un altre esdeveniment ("error").

Se sol·licita l'escriptura d'un mòdul que prenga com a base **zmq** (que podria anomenar-se **pzmq**). Aquest mòdul ha de proporcionar un objecte similar als del mòdul **zmq**. No obstant això, quan s'use el mètode **socket** sobre aquest objecte, s'haurà d'avaluar si el socket és de tipus **req**. Si ho és, el socket retornat ha d'admetre un mètode **request**() amb una signatura que admeta un nombre variable de segments en el missatge. Així:

```
var reply = sock.request(segment1, ..., segmentn);
```

On **reply** és una promesa que serà resolta quan el missatge de resposta arribe al socket **req** o quan un esdeveniment "error" siga generat.

En el primer cas, el valor de la promesa resolta és el missatge de resposta (com un vector de segments). En el segon cas la promesa serà rebutjada amb el valor de l'esdeveniment "error" generat pel socket **req** subjacent.

ACTIVITAT 3

OBJECTIU: Recuperar-se de fallades potencials en els socket **rep**.

ENUNCIAT: En l'activitat 1 es va observar que la fallada d'un servidor abans de contestar una petició pot bloquejar un client **req**.

En l'activitat 2 s'ha vist com convertir el patró d'intercanvi de missatges asíncron en un patró basat en promeses.

Aprofitarem l'après en l'activitat 2 per a manejar les fallades que bloquegen al procés client. A aquest efecte, se sol·licita el desenvolupament d'una extensió per al mòdul **pzmq**, on el mètode **request()** introduït en l'activitat 2 reba un paràmetre addicional: un timeout (especificat en segons). Ara la signatura del mètode serà:

var reply = sock.request(tmout, segment1, ..., segmentn);

El seu comportament ha de ser:

1. Si tmout==0 , llavors realitzarà la mateixa funció que en l'activitat 2.

2. En un altre cas, quan transcórreguen tmout segons sense que hi haja cap error ni s'haja rebut cap missatge, el mètode tancarà el socket i l'obrirà de nou de la mateixa manera, utilitzant els mateixos arguments que en la crida original. Addicionalment, la promesa reply serà rebutjada amb un valor "TMOUT".

Repetisca l'experiment de l'activitat 1 utilitzant la nova API. Descriga si el nou comportament és diferent a l'original. En aquest cas, descriga què ha millorat o empitjorat.

ACTIVITAT 4

OBJECTIU: Aplicar la tècnica de l'activitat 2 als sockets **dealer**. Organitzar de manera més senzilla les peticions concurrents mitjançant sockets **dealer**.

ENUNCIAT: Els sockets **dealer** poden ser utilitzats allí on els sockets **req** tinguen sentit. No obstant això, els sockets **req** utilitzen un segment delimitador en el missatge (tal com esperen els sockets **rep**), mentre que els sockets **dealer** no fan aquesta gestió.

En aquesta activitat se sol·licita una extensió de la funcionalitat del mòdul **pzmq** per a afegir un nou tipus de socket al que anomenarem **dealerReq**.

Els sockets **dealerReq** es comporten com una barreja entre **dealer** i **req**, però la seua funcionalitat s'implantarà mitjançant un mètode **request()** similar al de l'activitat 2, basat en els socket **dealer** de 0MQ.

Observe que els sockets **dealer** permeten l'enviament concurrent de missatges de petició (cosa que no ocorre en els sockets **req**). Per tant, la seua implementació ha de suportar això.

Pistes:

1. Tindrem només un socket **dealer** subjacent, que podrà enviar moltes peticions concurrents, fins i tot sobre sockets **rep** diferents. Com es podrà distingir a quina petició correspon cada resposta?

    a. Observe que amb sockets **req** això era senzill ja que a cada moment solament hi haurà una petició pendent. A aquesta petició estarà lligada la resposta actual. Amb un socket **dealer** això no es respectarà sempre.

    b. Observe que aquest problema ha de ser gestionat adequadament, ja que l'API facilitada "promet" la resposta per a una petició determinada… (no solament un esdeveniment "message" genèric del socket 0MQ).