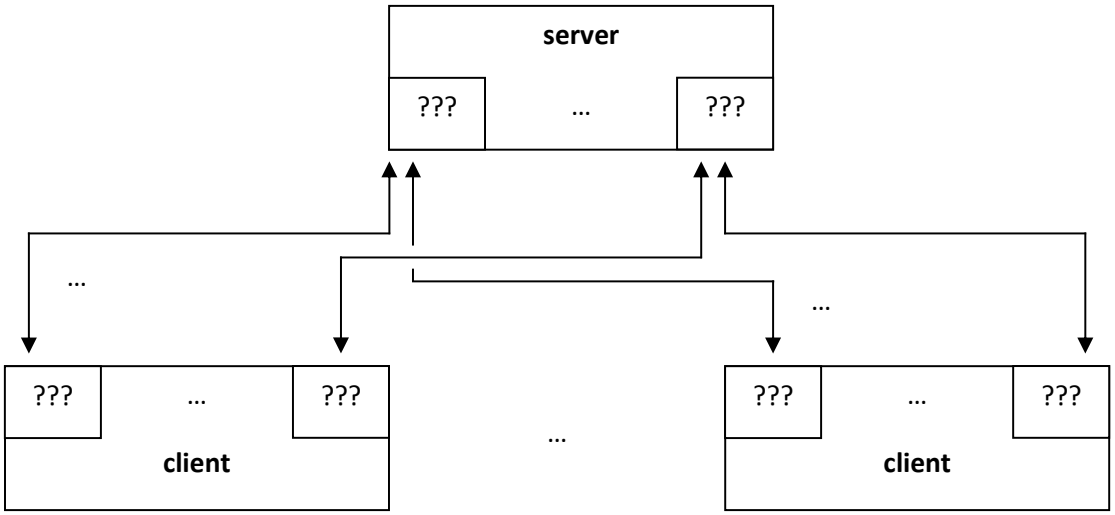
TSR: Seminari 3 – Activitats (II)

ACTIVITAT 1

Es desitja dissenyar un **servei de xat** usant 0MQ en l'enviament dels missatges entre els usuaris del servei. Per a fer això, cal dissenyar dos mòduls: un mòdul **servidor**, que distribuïsca els missatges entre els usuaris; i un mòdul **client**, que executarà cadascun dels usuaris que es connecte al servidor.

El funcionament dels mòduls ha d'ajustar-se al següent. Una vegada connectat al servidor, el mòdul client ha de poder enviar, en tot moment, missatges al servidor, amb la garantia que siguen rebuts per tots els altres clients que estiguen connectats. Per la seua banda, el servidor ha d'acceptar qualsevol missatge que li envie qualsevol client connectat, i retransmetre'l a tots els clients. El servidor també ha de notificar la incorporació d'un nou client als clients ja connectats, i la desconnexió d'un client als altres clients que romanguen.

En primer lloc, cal triar el nombre i tipus de **sockets 0MQ** que siguen més adequats per a satisfer les comunicacions requerides en el servei de xat. En la següent figura, els interrogants representarien sockets el tipus dels quals s'ha de decidir.



Una vegada definit el disseny dels mòduls, es pot començar amb la seua implantació.

En primer lloc, en el mòdul servidor, serà necessari declarar variables per als sockets triats i fer el seu *binding*, perquè estiguen disponibles per als clients. En el següent fragment de codi, es mostra l'estructura que tindria aquesta part del servidor (s'indiquen punts suspensius per a assenyalar codi a completar).

```
...

// *** chat server variables
var zmq = require('zmq')
  , ... = zmq.socket( ... )
  , ... = zmq.socket( ... )
  , ...

// *** binding sockets
... .bind('tcp://*:' + ... , function(err) {
  ...
})

... .bind('tcp://*:' + ... , function(err) {
  ...
})

...
```

Anàlogament, en el mòdul client, serà necessari declarar variables per als sockets triats i realitzar el seu *connect*, per a comunicar-se amb el servidor i, a través d'aquest, amb els altres clients. En el següent fragment de codi, es mostra l'estructura que tindria aquesta part del client (s'indiquen punts suspensius per a assenyalar codi a completar).

```
...

// *** chat client variables
var zmq      = require('zmq')
  , ...      = zmq.socket( ... )
  , ...      = zmq.socket( ... )
  , ...

// *** connecting sockets
... .connect( ... );
... .connect( ... );

...
```

A continuació, el client hauria d'enviar un missatge de sol·licitud de connexió al servidor (en el nostre cas d'estudi, aquesta sol·licitud serà acceptada sempre). En aquest missatge, el client proporcionaria el seu identificador, perquè les seues intervencions en el xat siguen reconegudes pels altres clients. Aquesta identitat podria facilitar-se com un argument en la invocació del mòdul client, o bé ser generada aleatòriament dins del propi mòdul. En el següent fragment de codi, es mostra l'estructura que tindria aquesta part del client (de nou, s'indiquen punts suspensius per a assenyalar codi a completar).

```
...

// *** setting client identity
// *** supposing a randNumber function that generates random numbers
if ( ... )  ... .identity = process.argv[ ... ];
else        ... .identity = 'client-' + randNumber( ... );

// *** sending connect message
var connect_msg = {'type': ... , 'identity': ... };
... .send( JSON.stringify(connect_msg) );

...
```

A partir d'aquest moment, el client hauria de poder enviar els missatges que volguera, així com rebre tots els missatges que altres clients enviaren. Per al segon cas (rebre missatges dels altres), hauria de connectar-se a l'escolta d'algun socket del servidor, i mostrar en la seua finestra terminal els missatges rebuts.

```
// *** receive everything
...

... .on('message', ...
  console.log( ... );
...
```

Per al primer (escriure i enviar les seues pròpies intervencions en el xat), el client hauria de tenir habilitada l'entrada estàndard, construir un missatge de xat que continguera, almenys, la seua identitat i el text de la seua intervenció, i enviar aquest missatge al servidor usant el socket que corresponga. A manera d'orientació:

```
// *** sending chat messages

process.stdin.resume();
process.stdin.setEncoding('utf8');

process.stdin.on('data', ...
  var chat_msg = { ... };
  ... .send( ... );
...
```

Finalment, el client hauria de poder notificar al servidor quan es desconnecta de la conversa. Per a fer això, s'ha d'aconseguir que, quan el client tanque l'entrada estàndard (prement Ctrl+D), s'envie un missatge especial de desconnexió al servidor. A més, en tancar l'aplicació client (prement Ctrl+C), s'haurien de tancar els sockets que el client haguera usat.

En el que concerneix al servidor, la seua tasca consisteix a estar a l'espera de missatges i retransmetre'ls adequadament. El servidor usarà algun socket per a mantenir-se a l'escolta de manera indefinida i, cada vegada que reba alguna cosa, ho identificarà, construirà el missatge que corresponga, i ho enviarà a tots els clients connectats. El servidor ha d'identificar qualsevol missatge rebut, atès que existiran 3 tipus de missatge:

- Missatge de connexió. És el missatge inicial que envia qualsevol client, informant de la seua identitat. Si, per exemple, el client s'identificara com "*Joan Ningú*", el servidor hauria d'enviar a tots els clients un missatge amb un text com el següent: "*Server > chat-client Joan Ningú connected!*".

- Missatge de xat. És el missatge que conté una intervenció en la conversa. El servidor ha de reenviar-ho a tots els clients, indicant la identitat de l'emissor. Si, per exemple, el client "*Joan Ningú*" escriu, en el seu missatge de xat, "*No tinc res a dir*", el servidor enviarà a tots un missatge amb un text com el següent: "*Joan Ningú > No tinc res a dir*".

- Missatge de desconnexió. És el missatge final que pot enviar qualsevol client si desitja avisar que abandona la conversa. El servidor hauria d'enviar a tots un missatge com el següent: "*Server > chat-client Joan Ningú disconnected!*".

Aleshores, a manera de plantilla d'aquesta part del servidor, consideren-se les següents línies:

```
// *** listening on input socket
// *** identifying message type
// *** broadcasting on output socket

... .on('message', function(msg) {
  var ...  = JSON.parse(msg)
    , type = ...
    , ...  , ...  , ...
  if ( type == ... )
    ...
  if ( type == ... )
    ...
  if ( type == ... )
    ...
  ... .send( ... );
})
```

Finalment, en tancar l'aplicació del servidor (prement Ctrl+C), s'haurien de tancar els sockets que el servidor haguera usat.

ACTIVITAT 2

Es desitja dissenyar un **servei de descàrrega de fitxers de text** usant 0MQ en l'enviament dels missatges entre el servidor de fitxers i els usuaris del servei. Per a fer això, cal dissenyar dos mòduls: un mòdul **servidor**, que reba les peticions de descàrrega que envien els usuaris connectats, i les atenga, enviant els fitxers sol·licitats; i un mòdul **client**, que executarà cadascun dels usuaris que es connecte al servidor.

El funcionament dels mòduls ha d'ajustar-se al següent:

- El servidor atendrà cada petició. Si el fitxer sol·licitat no existeix, contestarà amb algun missatge especial (per exemple, un missatge buit). Si el fitxer existeix, el llegirà. El fitxer llegit no serà enviat com a contingut d'un únic missatge. El servidor dividirà el contingut en blocs de 1K, i enviarà tants missatges com siguen necessaris, en funció de la grandària del fitxer (tots els missatges contindran 1K de dades, excepte l'últim que contindrà la resta).

- Una vegada connectat al servidor, el mòdul client podrà escriure en l'entrada estàndard el nom del fitxer que desitja descarregar, i quedarà a l'espera que la descàrrega es complete (o s'informe d'algun error: per exemple, inexistència del fitxer sol·licitat), després d'això hauria de poder repetir l'operativa d'indicar nom de fitxer i esperar la seua descàrrega tantes vegades com vulga. Donada la manera d'enviament del fitxer per part del servidor, el mòdul client haurà d'anar escrivint els blocs de 1K (transmesos en cada missatge) en el fitxer local associat a la descàrrega en curs, i procedirà a tancar el fitxer quan reba l'últim missatge associat al mateix.

A més, en el disseny i implementació del servei, es tindran en compte les següents consideracions:

- Per a simplificar la gestió del servei, se suposarà que qualsevol client ja coneix els noms dels fitxers disponibles en el servidor.

- Els clients sol·licitaran fitxers usant noms de ruta relatius al directori on estiga el servidor. Així, si un client sol·licita el fitxer "*kipling.txt*", aquest fitxer hauria d'existir en el directori on estiga el mòdul servidor; i si sol·licitara el fitxer "*textos/kipling.txt*", el servidor cercaria el fitxer "*kipling.txt*" en el subdirectori "*textos*".

- L'únic tipus d'error que comprovarà el servidor és la petició de descàrrega relativa a un fitxer inexistent (responent al client, com s'ha indicat abans, amb algun missatge especial).

- En aquest senzill servei de descàrrega, se suposarà que no es presentaran altres possibles errors (com a denegació d'accés al fitxer sol·licitat, sol·licitud d'un fitxer que no siga de text, format incorrecte del missatge de petició del client, etc.). Per tant, el mòdul servidor no haurà de fer cap tractament d'aquests errors. En una aplicació més completa i funcional, aquestes excepcions haurien de tractar-se.

ACTIVITAT 3

Es desitja dissenyar un **servei de còmput de funcions matemàtiques** usant 0MQ en l'enviament dels missatges entre els servidors de còmput i els usuaris o clients del servei, mitjançant una arquitectura centralitzada en un broker. Per a fer això, cal dissenyar tres mòduls:

- Un mòdul **broker** que, d'una banda, reba les peticions dels clients i reenvie aquestes peticions, de manera equitativa, als servidors de còmput i, d'altra banda, reba els resultats dels càlculs i els reenvie als clients que els van sol·licitar. El funcionament del servei només requerirà l'execució d'un mòdul broker.

- Un mòdul **servidor** que, a través del broker, reba les peticions de còmput dels clients, realitze els càlculs sol·licitats i envie, també a través del broker, els resultats als clients. El servei ha de dissenyar-se de manera que puga funcionar amb qualsevol nombre de servidors.

- Un mòdul **client**, que executarà cadascun dels usuaris que es connecte al broker. Aquest mòdul enviarà les peticions de càlcul d'un usuari i quedarà a l'espera de rebre els missatges corresponents als resultats. El servei ha de dissenyar-se de manera que puga funcionar amb qualsevol nombre d'usuaris connectats.

Per a simplificar el problema, es considera que el còmput de les funcions matemàtiques es limita a les incloses en el següent mòdul, **myMath.js**. Aquest mòdul haurà d'usar-se en la resolució de l'exercici, i no haurà de modificar-se.

```
1:  function fibo(n) {
2:     return (n<2) ? 1 : fibo(n-2) + fibo(n-1)
3:  }
4:
5:  function fact(n) {
6:     return (n<2) ? 1 : n * fact(n-1)
7:  }
8:
9:  function prim(n) {
10:    for (var i=2; i<=Math.sqrt(n); i++)
11:       if (n%i === 0) return false;
12:    return true;
13: }
14:
15: function calculate(f, n) {
16:   var res = 0;
17:   switch (f) {
18:     case 'fibo': res = fibo(n); break;
19:     case 'fact': res = fact(n); break;
20:     case 'prim': res = prim(n); break;
21:     case 'sin':  res = Math.sin(n); break;
22:     case 'cos':  res = Math.cos(n); break;
23:     case 'tan':  res = Math.tan(n); break;
24:     case 'log':  res = Math.log(n); break;
25:     case 'exp':  res = Math.exp(n); break;
26:     case 'arrel': res = Math.sqrt(n); break;
27:     default:     res = NaN;
28:   }
29:   return res;
30: }
31:
32: ids = ['fibo', 'fact', 'prim', 'sin', 'cos', 'tan',
```

```
33:        'log', 'exp', 'arrel', 'strange'];
34:
35: exports.eval = calculate;
36: exports.funcs = ids;
```

El mòdul **myMath** exporta una funció i un vector. Els mòduls client i servidor hauran d'importar el mòdul. En concret, la funció **eval** serà usada pel mòdul servidor per a calcular les funcions sol·licitades pels clients, i el vector **funcs** serà usat pel mòdul client per a especificar la funció que es demana calcular.

A més, el funcionament dels mòduls ha d'ajustar-se al següent:

- El **broker** disposarà de dos ports. Un port servirà per a comunicar-se amb els usuaris del servei (mòduls client), l'altre port per a comunicar-se amb els servidors de còmput. Qualsevol petició d'un client serà redirigida cap a algun dels servidors connectats, seguint un repartiment equitatiu de la càrrega (estratègia round-robin). Qualsevol resposta d'un servidor serà reexpedida al client que la va sol·licitar. El broker serà un mòdul que s'execute indefinidament.

- El **servidor** disposarà d'un port on el broker li farà arribar peticions de còmput i per on enviarà les respostes corresponents. El servidor farà el càlcul indicat en la petició rebuda i construirà un missatge de resposta en el qual inclourà, almenys, el resultat del càlcul. Atès que el càlcul de qualsevol funció del mòdul **myMath** és immediat (amb excepció de la funció **fibo** per a nombres majors a 40), se suggereix usar una temporització (amb la funció **setTimeout**) en l'enviament dels missatges de resposta del servidor. Així, la simulació plantejada en aquest exercici serà una mica més realista. El servidor serà un mòdul que s'execute indefinidament.

- El **client** disposarà d'un port per a comunicar-se amb el broker, per on enviarà les seues peticions i per on esperarà les respostes. El missatge de petició haurà d'incloure, almenys, un identificador de funció a avaluar i un valor (l'argument per a la funció). Els identificadors de funció seran, obligatòriament, valors del vector **funcs**, importat del mòdul **myMath**[1]. Se suggereix implantar el client de manera que llance automàticament un cert nombre de peticions[2] i espere la recepció de les respostes a totes aquestes peticions, mostrant els resultats en l'eixida estàndard. Si se segueix aquest suggeriment, el client acabarà la seua execució quan reba i mostre totes les respostes.

[1] L'última component de l'array **funcs**, l'identificador **strange**, pretén representar una funció que no pot avaluar el servidor de còmput. Aleshores, per a qualsevol valor **n**, el resultat d'eval(strange, **n)** és **NaN**, Not a Number.

[2] Es poden triar aleatòriament tant les funcions a avaluar com els arguments d'aquestes funcions, si bé és recomanable no invocar la funció **fibo** amb un valor major a 40, donat el cost exponencial que té la implementació recursiva de la funció.

ACTIVITAT 4

Aquesta activitat és una ampliació o millora del **servei de còmput de funcions matemàtiques** desenvolupat en l'activitat anterior. Es manté l'arquitectura centralitzada en un broker que permet interconnectar un nombre variable tant de mòduls client com de mòduls servidors o treballadors.

La diferència resideix en el fet que hi haurà dos tipus de servidors. Existiran servidors limitats en les seues prestacions, que no seran adequats per a avaluar les funcions costoses (en el nostre cas, les funcions **fibo**, **fact** i **prim** del mòdul **myMath**). I existiran servidors potents, capaços de processar en un temps raonable qualsevol funció. Perquè el broker puga saber el tipus de cada servidor connectat, aquest haurà d'enviar un missatge especial de connexió al broker en el qual incloga el seu identificador i algun valor de tipus *flag* que especifique si és un servidor limitat o potent.

En els missatges de peticions dels clients, a més d'incloure l'identificador de funció i el seu argument, es podria incloure també un valor *flag* que especificara si la funció és costosa (és a dir, si és **fibo**, **fact** o **prim**) o si no ho és (qualsevol de les altres, importades del mòdul **Math**). Per a poder redirigir adequadament les peticions cap als servidors, el broker hauria de disposar d'alguna estructura de dades interna on mantinguera les identitats i tipus dels servidors connectats.

Si la funció a avaluar en una petició és costosa, el broker enviaria la petició cap a algun dels servidors potents connectats. Si la funció a avaluar en una petició no és costosa, el broker podria enviar la petició cap a qualsevol dels servidors connectats. Feta aquesta distinció, el broker hauria de procurar mantenir el repartiment equitatiu de la càrrega. Aleshores, l'avaluació de les funcions “lleugeres en còmput” hauria de repartir-se equilibradament entre tots els servidors, i l'avaluació de les funcions “pesades en còmput” hauria de repartir-se equilibradament entre els servidors potents.

Les modificacions plantejades en aquesta activitat, respecte a l'anterior, podrien implicar canvis en els patrons de comunicació (tipus de sockets 0MQ a usar) i en l'estructura dels missatges a construir i transmetre (missatges multi-part). Igual que en les altres activitats d'aquest butlletí, abans de posar-se a implementar res, és molt recomanable analitzar el problema plantejat i prendre unes decisions de disseny adequades.