TSR: Exercici 3

Aquests programes són un client i un servidor. El servidor rep un argument des de la línia d'ordres i retorna una resposta que inclou dos segments. El primer manté el contingut de la sol·licitud i el segon proporciona la resposta. La resposta és el valor de la sol·licitud multiplicat pel valor rebut com a argument des de la línia d'ordres. Una vegada el client rep cada resposta, mostra el contingut del missatge.

Act3-client.js

```
var zmq = require('zmq');
var req = zmq.socket('req');
var counter = 1;
const interval = 1000;

req.bindSync('tcp://*:8000');

function sendMsg() {
  req.send(counter++);
}

function handler(segm1,segm2) {
  console.log('Request: %d, reply: %s',
              segm1,segm2);
  if (segm1>99) process.exit(0);
}

req.on('message', handler);

setInterval(sendMsg, interval);
```

Act3-server.js

```
var zmq = require('zmq');
var rep = zmq.socket('rep');
var args = process.argv.slice(2);
if (args.length < 1) {
  console.error('Please provide an argument!');
  process.exit(1);
}
rep.connect('tcp://127.0.0.1:8000');

rep.on('message', function(msg) {
  rep.send([msg,msg*args[0]]);
});
```

Responga justificadament les següents preguntes en una fulla a part:

1. Justifique si aquests programes desenvolupen correctament el que s'ha descrit en l'enunciat. En cas que no siga així, explique quines sentències són incorrectes i proporcione una versió correcta d'elles. (1 punt)
2. Explique quants missatges són enviats pel client i amb quina ràtio d'enviament (és a dir, quants missatges en cada unitat de temps). (1 punt)
3. Descriga si existeix algun límit per al nombre de clients que podran ser iniciats. (1 punt)
4. Descriga si existeix algun límit per al nombre de servidors que podran ser iniciats. (1 punt)
5. Justifique si, amb el codi mostrat a dalt, clients i servidors poden ser situats en ordinadors diferents. (1 punt)
6. Descriga què serà mostrat pel client durant els seus primers 10 segons d'execució si aquesta seqüència d'accions ocorre en un ordinador determinat: (1) Un client és iniciat en l'instant 0 seg.; (2) en l'instant 2.9 seg., un servidor és iniciat utilitzant "**node Act3-server 2**"; (3) en l'instant 5.2 seg., un segon servidor és iniciat utilitzant "**node Act3-server 5**". (3 punts)
7. Explique com s'hauran d'iniciar múltiples servidors per a generar la següent eixida en el client (2 punts):

```
Request: 1, reply: 3
Request: 2, reply: 6
Request: 3, reply: 9
Request: 4, reply: 8
Request: 5, reply: 25
Request: 6, reply: 18
Request: 7, reply: 14
Request: 8, reply: 40
Request: 9, reply: 27
Request: 10, reply: 20
...
```

Solució comentada

Solucions

1. Tots dos programes desenvolupen correctament el que s'ha descrit en l'enunciat. Act3-server.js deixa els arguments rebuts en la variable **args**. En el bloc entre les línies 4 i 7 comprova si s'ha rebut almenys un argument i mostra un missatge d'error en cas que no siga així. En aquest mateix bloc, mitjançant **process.exit(1)**, es tanca el procés. La resposta construïda pel servidor es genera en l'última sentència del programa. Allí es passa com a argument de la crida al mètode **send()** un vector amb dos components. Cada component serà un segment del missatge de resposta. El primer segment és el valor rebut (observe's que és l'argument del *listener* per a l'esdeveniment **'message'** del socket utilitzat pel servidor) i el segon s'obté multiplicant el primer segment pel valor rebut com a argument des de la línia d'ordres. El comportament del client es descriu amb deteniment en la resposta de la següent pregunta. També implanta allò que es demanava en l'enunciat.
2. Amb l'última sentència del programa client s'estableix que la funció **sendMsg** serà invocada una vegada per segon. Dins de la funció **sendMsg** s'observa que en cada invocació s'incrementa el valor d'un comptador. Aquest mateix valor, abans de l'increment, és enviat com a contingut d'un missatge al servidor. Posteriorment, en la funció **handler**, observem que es mostra un missatge per pantalla, reportant quin ha sigut el valor enviat i quin el valor obtingut com a resposta. Si el valor del primer segment supera 99 unitats, el client acaba. En estar utilitzant un patró REQ-REP això implica que el client generarà 100 missatges i abans d'emetre el centè primer (ja que la resposta al centè arriba abans), finalitzarà.
3. El client utilitza **bindSync** emprant un nombre de port fix (el 8000). Com no poden realitzar-se dues crides **bind** amb èxit sobre un mateix port local, només es podrà generar un únic client.
4. El servidor es connecta al port utilitzat pel client en la màquina local. No hi ha res que impedixca que diversos processos es connecten a un mateix URL. Per tant, no existeix cap límit sobre el nombre de servidors a iniciar.
5. No poden situar-se en ordinadors diferents ja que el servidor utilitza un URL local per a connectar-se. Client i servidor han d'estar en un mateix ordinador per a poder interactuar.
6. El client genera un missatge per segon. Això implica que en l'instant 1seg s'enviarà la primera sol·licitud al servidor o servidors. En aquest moment encara no hi ha cap servidor connectat, per la qual cosa el missatge de petició es queda en el buffer d'eixida. Ocorre el mateix en l'instant 2 seg. En l'instant 2.9 seg. es connecta el primer servidor. Amb això, el primer missatge arriba al servidor i és respost. Com tots dos processos estan en la mateixa màquina, costarà uns pocs mil·lisegons (encara que dependrà de la càrrega que hi haja en l'equip i l'estratègia de planificació utilitzada pel sistema operatiu) entregar la resposta. En aqueix moment es mostrarà "**Request: 1, reply: 2**" en pantalla. En haver-se lliurat la primera resposta, el socket REQ del client serà capaç de transmetre el segon missatge que es va deixar en la seua cua d'eixida. Als pocs mil·lisegons es mostrarà "**Request: 2, reply: 4**" en pantalla. En els instants 3 seg, 4 seg i 5 seg es mostraran respectivament els missatges "**Request: 3, reply: 6**", "**Request: 4, reply: 8**" i "**Request: 5, reply: 10**". A partir del segon 6 tenim dos servidors connectats al socket REQ del client. Quan això ocorre, el socket REQ distribueix els missatges enviats circularment entre els processos connectats a ell. Per això, s'observarà el següent en els cinc missatges que faltava distribuir en els segons 6, 7, 8, 9 i 10: "**Request: 6, reply: 12**" (multiplicat per 2), "**Request: 7, reply: 35**" (multiplicat per 5), "**Request: 8, reply: 16**" (multiplicat per 2), "**Request: 9, reply: 45**" (multiplicat per 5) i "**Request: 10, reply: 20**" (multiplicat per 2).
7. Haurem de començar observant quin factor s'ha utilitzat per a realitzar cada multiplicació. Així, veiem que els missatges 1, 2, 3, 6 i 9 s'han multiplicat per 3, mentre que els missatges 4, 7 i 10 s'han

multiplicat per 2 i els missatges 5 i 8 s'han multiplicat per 5. Per tant, al final hi havia tres servidors, però els tres primers missatges van ser atesos per un mateix servidor (el que multiplicava per 3).
Hi ha diverses formes de generar aquest resultat. Comentarem algunes de les possibles.
Un possible exemple seria:

- Entre els instants 0 seg i 3.5 segons s'ha hagut de llançar amb "**node Act3-server 3**" un primer servidor que multiplicava les sol·licituds per 3. Si el servidor es va llançar a partir dels tres segons, ha hagut de tenir temps per a ser l'únic servidor existent capaç d'atendre els tres primers missatges mantinguts en la cua d'eixida del client.
- Abans de l'instant 4" però després de l'instant 2.1", respectant les restriccions comentades en el punt anterior, s'ha generat un altre servidor amb l'ordre "**node Act3-server 2**". Amb això, aquest segon servidor ha sigut capaç de rebre el quart missatge emès pel client.
- Abans de l'instant 5" i després d'haver llançat el segon servidor, s'ha arribat a llançar un tercer servidor amb l'ordre "**node Act3-server 5**". Per exemple, tant el segon com el tercer servidor van poder ser llançats de manera consecutiva poc després dels 2" d'haver iniciat el client, si assumim que el primer servidor va ser llançat abans dels 2".