TSR - Pràctica 2: **0MQ**

Curs 2022/23

Contents

1. Introducció

1.1. Objectius

- Afermar els conceptes teòrics introduïts en el tema 3
- Experimentar amb diferents patrons de disseny (patrons bàsics de comunicació) i tipus de sockets
- Aprofundir en el patró broker (proxy invers)

1.2. Proposta per a l'organització del temps

- Sessió 1.- Prova dels patrons bàsics i l'aplicació Xat
- Sessió 2.- Publicador rotatori, prova del patró broker i estadístiques broker
- Sessió 3.- Broker per a clients + broker per a workers
- Sessió 4.- Prova del patró broker tolerant a fallades

1.3. Mètode de treball

- Per simplicitat, llancem els diferents components de cada aplicació en la mateixa màquina (IP = localhost)

  - Però també es poden repartir els components sobre màquines diferents

- No has de modificar el codi proporcionat ni la seua estructura de directoris
  - Els programes requereixen arguments en línia d'ordres
  - Aquests arguments permeten plantejar diferents escenaris
  - Pots modificar els números de port dels exemples (pregunta al professor)

- Es recomana utilitzar el fitxer `refZMQ.pdf` com a referència
- En les diferents tasques es plantegen qüestions que l'alumne ha de respondre
  - Permeten verificar que s'han entès els conceptes bàsics
  - Has d'esbrinar la resposta i comprendre la justificació d'aquesta resposta

- El fitxer `fuentes.zip` (en PoliformaT) conté el codi per a fer les diferents proves

2. Tasques

2.1. Prova dels patrons bàsics i l'aplicació xat

- Tots els components utilitzats en aquestes proves utilitzen la biblioteca `tsr.js`, que es descriu en el subapartat corresponent
- En cada sub-apartat es detalla un dels patrons bàsics. Els diferents components d'un patró estan reunits en un mateix directori
- Per a realitzar les proves sobre un patró hem d'obrir diversos terminals (entre 2 i 5 segons la prova a fer), situar-nos en cada terminal sobre el directori corresponent al patró, i executar l'ordre que s'indica per a cada terminal

2.1.1. Biblioteca `tsr.js`

- Per a simplificar les proves dels diferents patrons i el codi de cada component suposem definida la següent biblioteca `tsr.js`, que importa la biblioteca `zeromq` i exporta les funcions:
  - `error(msg)` Mostra el missatge d'error i finalitza l'execució del programa
  - `lineaOrdenes(params)` Comprova que la línia d'ordres conté els paràmetres indicats, i crea les variables corresponents
  - `traza(f,noms,valors)` Mostra el valor dels arguments quan s'invoca la funció f
  - `adios(sockets, comiat)` Tanca els sockets indicats. mostra el missatge de comiat, i finalitza el programa
  - `creaPuntoConexion(socket,port)` Aplica l'operació `bind` sobre el port indicat, i verifica el resultat
  - `conecta(socket,ip,port)` Aplica l'operació `connect` sobre `tcp://ip:port`

2.1.2. Prova del patró client/servidor (`req/rep`)

Ens situem en el directori `req-rep`, on trobem els fitxers `cliente1.js`, `cliente2.js` i `servidor.js`

- Un client i un servidor
  - terminal 1) `node servidor.js A 9990 2 Hola`
  - terminal 2) `node cliente1.js localhost 9990 Pepe`
  - Preguntes
    - Què ocorre si passem un nombre d'arguments incorrecte? i si estan fora d'ordre?

    * Comprova si l'ordre en què arranquem els components afecta al resultat. Indica la raó
    * En relació amb els missatges multi-segment:
        · De quina forma construeix l'emissor un missatge multi-segment?
        · Com accedeix el receptor als diferents segments del missatge?
    * El client finalitza després de rebre la resposta a la quarta petició. Quan acaba el servidor?

- Un client i dos servidors
  - terminal 1) `node servidor.js A 9990 2 Hola`
  - terminal 2) `node servidor.js B 9991 2 Hello`
  - terminal 3) `node cliente2.js localhost 9990 localhost 9991 Pepe`
  - Preguntes
    * Comprova si l'ordre d'arrancada afecta al resultat. Indica la raó
    * Què ocorre si tots dos servidors reben el mateix número de port?
    * Què ocorre si els dos servidors reben un valor de segons diferent (ex. 1 i 3 respectivament)? Afecta a l'ordre en què es respon al client?
    * La pràctica totalitat del temps el consumeixen els servidors. Aconseguim reduir a la meitat el temps d'execució del client en utilitzar 2 servidors?
    * Si volem usar 3 servidors, cal modificar el codi del client?
    * Amb un nombre de peticions parell, podem garantir que cada servidor atén la mateixa quantitat de peticions?

- Dos clients i un servidor
  - terminal 1) `node servidor.js A 9990 2 Hola`
  - terminal 2) `node cliente1.js localhost 9990 Pepe`
  - terminal 3) `node cliente1.js localhost 9990 Ana`
  - Preguntes
    * Comprova si l'ordre en què arranquem els components afecta al resultat. Indica la raó
    * Podem assegurar que cada client rep únicament les respostes a les seues pròpies peticions? Indica la raó
    * En cas de plantejar una quantitat diferent de clients (ex. 3), seria necessari modificar el codi del client o del servidor?
    * En cas que un dels clients acabe abans d'hora (ctrl-C), l'altre continua rebent respostes? Indica la raó

2.1.3. Prova del patró pipeline (push/pull)

- Ens situem en el directori `push-pull`, on trobaràs els fitxers `origen1.js`, `origen2.js`, `filtro.js` i `destino.js`

- origen1 -> destino A-B
  - terminal 1) `node origen1.js A localhost 9000`
  - Preguntes

    * Comprova si l'ordre en què arranquem els components afecta al resultat. Indica la raó
    * Indica la raó per la qual el socket definit en origen.js no defineix socket.on('message',..)

- origen1 -> filtro -> destino A-B-C
  - terminal 1) `node origen1.js A localhost 9000`
  - terminal 2) `node filtro.js B 9000 localhost 9001 2`
  - terminal 3) `node destino.js C 9001`
  - Preguntes
    * Comprova si l'ordre en què arranquem els components afecta al resultat. Indica la raó
    * Indica la raó per la qual origen.js i destino.js defineixen un únic socket cadascun, però filtro.js defineix 2 sockets
    * Si `origen` genera 4 missatges i `filtro` retarda 2 segons, quant creus que tarda l'últim missatge d'`origen` a arribar a `destino`?

- origen2 -> [filtro, filtro] -> destino A-(B,C)-D
  - terminal 1) `node origen2.js A localhost 9000 localhost 9001`
  - terminal 2) `node filtro.js B 9000 localhost 9002 2`
  - terminal 3) `node filtro.js C 9001 localhost 9002 3`
  - terminal 4) `node destino.js D 9002`
  - Preguntes
    * Comprova si l'ordre en què arranquem els components afecta al resultat. Indica la raó
    * Com es reparteix el lliurament de missatges als filtres B i C?
    * Poden treballar B i C en paral · lel (ex. si s'executaren en màquines diferents)?
    * En quin ordre arriben els missatges a `destino`? Com afectaria el comportament modificar els segons del filtre C?

2.1.4. Prova del patró difusió (pub/sub)

Ens situem en el directori `pub-sub`, on trobem els fitxers publicador.js i suscriptor.js

- terminal 1) `node publicador.js 9990 Economia Esports Cultura`
- terminal 2) `node suscriptor.js A localhost 9990 Economia`
- terminal 3) `node suscriptor.js B localhost 9990 Esports`
- terminal 4) `node suscriptor.js C localhost 9990 Economia`
- Preguntes
  - Indica si l'ordre en què arranquem els components afecta al resultat
  - Què passa amb els missatges de Cultura?
  - Pot rebre el mateix missatge més d'un subscriptor?
  - Com es pot canviar la quantitat de missatges que genera el publicador?
  - Els subscriptors no acaben. Pensa en una modificació perquè acaben després d'un temps determinat sense rebre missatges
  - És possible que el publicador genere algun missatge quan encara no ha processat les connexions dels subscriptors. Què passa amb aqueixos missatges?

2.2. Prova aplicació xat

En el directori `chat` trobaràs la implementació d'una aplicació xat rudimentària. Analitza el codi i comprova el seu funcionament.

- terminal 1) `node servidorChat.js 9000 9001`
- terminal 2) `node clienteChat.js Pepe localhost 9000 9001`
- terminal 3) `node clienteChat.js Ana localhost 9000 9001`
- Preguntes
  - En què afecta l'ordre en què arranquem els components?
  - Indica la raó per la qual tots dos punts de connexió es creen en el servidor
  - El servidor no manté una llista de clients connectats. Per quina raó?
  - Pensa com modificar un client perquè atenga únicament missatges d'alguns temes concrets

2.3. Publicador rotatori

Utilitzant el patró `pub/sub`, desenvolupa un programa `publicador.js` que s'invoca com

```
node publicador.js port numMissatges tema1 tema2 tema3 ...
```

on:

- `port` = el port al qual han de connectar-se els subscriptors (l'ordenador és `localhost`)
- `numMissatges` = nombre de missatges a emetre, després de la qual cosa el publicador acaba
- `tema1 tema2 tema3 ...` = nombre variable de temes (a priori no sabem quants)

Ha de generar un missatge per segon amb l'estructura `[tema, numMissatge, numRonda]`

- tema (en ordre circular)
- número de missatge
- ronda (primera iteració sobre els temes, segona, etc)

Exemple:

```
node publicador 8888 5 Politica Futbol Cultura
```

ha d'emetre (cada missatge en un segon):

```
Politica 1 1
Futbol 2 1
Cultura 3 1
Politica 4 2
Futbol 5 2
```

2.4. Tasques sobre el patró broker/Workers

2.4.1. Prova del patró broker/workers

En el directori `broker-worker` trobaràs diversos fitxers, però per a aquesta pràctica únicament utilitzem `cliente.js`, `brokerRouterRouter.js`, i `workerReq.js` Analitza el codi i comprova el seu funcionament:

- terminal 1) `node brokerRouterRouter 9000 9001`
- terminal 2) `node workerReq w1 localhost 9001`
- terminal 3) `node workerReq w2 localhost 9001`

- terminal 4) `node cliente A localhost 9000`
- terminal 5) `node cliente B localhost 9000`
- Preguntes:
  - Com afecta al resultat canviar l'ordre d'arrancada dels workers (terminals 2 i 3)? I dels clients (terminals 4,5)?
  - Què passa si arranquem el broker al final (el 1) passa al 5))

2.4.2. Estadístiques broker

Modifica el codi de `brokerRouterRouter.js` per a mantenir el total de peticions ateses i el nombre de peticions ateses per cada worker

- El broker ha de mostrar aquesta informació en pantalla cada 5 segons
- Preguntes
  - Indica una estratègia per a mantenir en el broker estadístiques separades per a cada worker
  - Indica com aconseguir que s'execute una acció de forma repetida (periòdica)
  - Si arriba una petició i li la passem al worker w, hem d'incrementar el nombre de peticions ateses per w (i el total) en aqueix moment, o quan arribe la resposta des de w?

2.4.3. Broker per a clients + Broker per a workers

Prenem com a punt de partida el fitxer `brokerRputerRouter.js` disponible en `fuentes.zip` directori `broker-workers`

- Reescriu el codi per a tenir dos brokers interconnectats entre si als quals anomenem `broker1.js` i `broker2.js`
- La solució triada ha de mantenir les mateixes característiques externes (és a dir, enfront de clients i enfront de workers) que el codi original

Concretament:

- `broker1` coneix als clients, i manté la cua de peticions pendents
- `broker2` coneix als brokers, i es responsabilitza de mantenir els workers disponibles i repartir la càrrega
- Tot client envia la seua petició a `broker1`, que la guarda en la cua de peticions pendents o la passa a `broker2`
  - NOTA.- `broker1` no sap quins workers estan disponibles, però podem organitzar el codi perquè sàpia quants estan disponibles
- Quan arriba una petició a `broker2`, aquest l'envia al worker corresponent
- La resposta del worker arriba a `broker2`, que la passa a `broker1`, i aquest al client
- L'alta d'un worker arriba a `broker2` (i si es considera necessari es pot informar a `broker1`)

Has de decidir com es comuniquen broker1 i broker2 (hi ha més d'una alternativa)

- Preguntes
  - Quines alternatives podem usar per a comunicar entre sí els dos brokers?
  - És convenient avisar a broker1 quan es dona d'alta un worker?
  - És convenient avisar a broker2 quan arriba una petició i no hi ha workers disponibles?

2.4.4. Broker tolerant a fallades de workers

En la carpeta `brokerToleranteFallos` hi ha una nova versió del fitxer `broker.js` que pot gestionar algunes situacions de fallada.

Provem el broker 'normal'. Ens situem en el directori broker-worker i llancem

- terminal 1) `node brokerRouterRouter 9000 9001`
- terminal 2) `node workerReq w1 localhost 9001`
- terminal 3) `node workerReq w2 localhost 9001`
- terminal 4) `node workerReq w3 localhost 9001`
- terminal 2) `ctrl-C`
- terminal 5) `node cliente A localhost 9000 & node cliente B localhost 9000 & node cliente C localhost 9000`
- Preguntes
  - Quantes respostes s'obtenen? Indica quins treballadors les han enviades
  - Queden clients esperant?

Repeteix la prova, però aquesta vegada ens situem en el directori `brokerToleranteFallos`

- terminal 1) `node broker.js 9000 9001`
- la resta igual que en la prova anterior
- Preguntes
  - Queden clients esperant?
  - El tancament (caiguda) del worker és transparent per al client?
  - Únicament s'aborda possibles fallades de workers. Indica si es pot aplicar alguna estratègia davant una possible fallada del broker