Tema 3.- Middleware. ZeroMQ

TSR 2020. Juansa Sendra, grup A

Introducció

- Desenvolupem components d'un sistema distribuït $\rightarrow$ han de cooperar entre ells
  - Ex. servei planificació de rutes: depén de servei GIS (informació distàncies)
  - Ex. sistema d'autorització: necessita servei de reconeixement biomètric
- Complicat
  - Poden desenvoluparse per programadors diferents, amb entorns de programació diferents
  - Molts detalls a considerar, especialment per a sol.licitar serveis
    - Com localitzar al servidor correcte? quina es l'API de un servei? com construir/interpretar peticions de servei? com associar a client la resposta correcta? ...
  - Depuració complexa
  - Seguretat?
  - Gestió de fallades?

Introducció.- Solucions per a reduir la complexitat

- Utilització d'estàndards
  - Faciliten la interoperabilitat
  - Introdueixen formes racionals de fer les coses
  - Proporcionen funcionalitat d'alt nivell
- Reutilització de solucions o components previs
  - Menys codi que escriure
  - Més garanties
- Middleware
  - Nivell de programari i serveis entre les aplicacions i el SO
  - Introdueix transparència de ... (ubicación, replicació, fallades, ...)

Middleware

- Perspectiva del programador
  - Implantació senzilla (conceptes clars i ben definits)
  - Resultat fiable (proporciona metodologia de desenvolupament estandarditzada i ben definida)
  - Simplifica manteniment (revisions APIs)
- Perspectiva del administrador
  - Simplifica instal.lació, configuració, actualització
  - Facilita interoperabilitat (amb productes de terceres parts)

Middleware orientat a comunicacions (Sistemes de missatgeria)

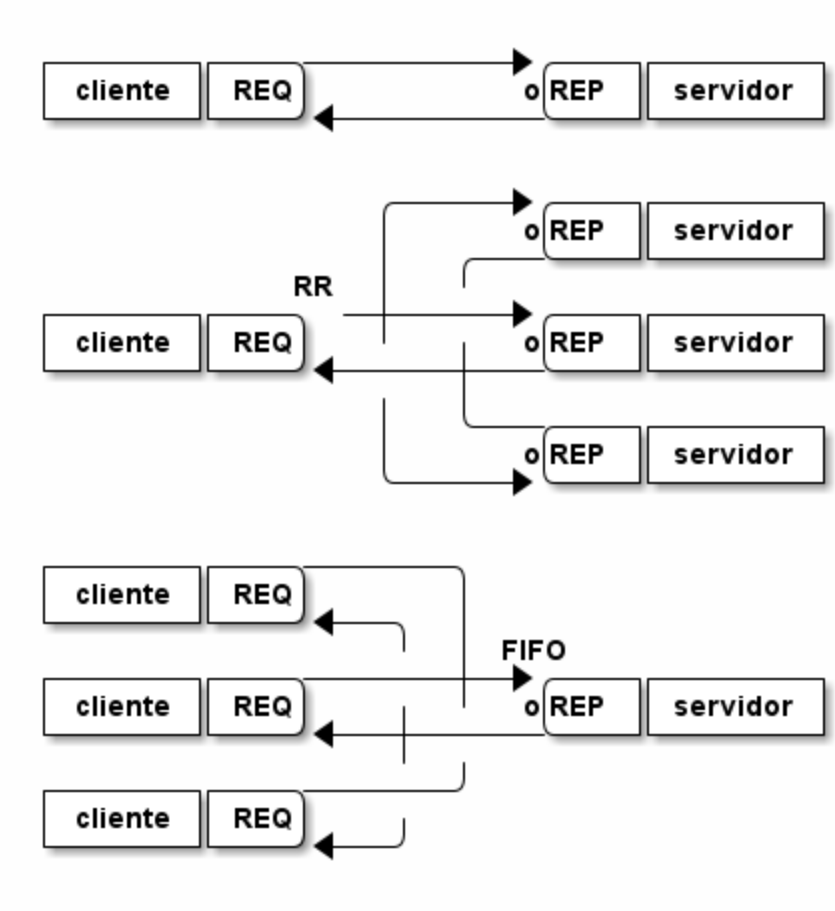
- Comunicació flexible
  - Transmisió atòmica (tot o res)
  - Grandaria arbitraria, missatges estructurats
  - Gestió de cues, amb certes garanties d'ordre
- No imposa visió d'estat compartit
  - L'estat compartit escala malament i pot provocar problemes de concurrència
  - Acoblament feble
- Encaixa amb el model conduït per esdeveniments
  - Implicitament asincrònic (desacobla emissor/receptor)
- Classificació del sistemes de missatgeria
  - Persistents.- buffers per a missatges. No requereix receptor actiu
  - No persistents.- el receptor ha de estar actiu per a transmetre el missatge
    - Amb gestor (ex. AMQP, JMS), o sense (ex. 0MQ) $\downarrow$ garanties $\uparrow$ escalable.

ZeroMQ.- Middleware de comunicacions ...

- Simple
  - URLs per a nomenar 'endpoints'
  - API semblant als sockets BSD (familiar): bind/connect, send/receive, ..
    - Molts tipus de sockets, per a implantar diferents patrons de comunicació
  - Només es una biblioteca (no tenim que arrancar cap servidor, etc.)
    - instalar: `npm install zeromq@5` utilitzar: `const zmq = require('zeromq')`
  - Model Entrada/eixida asincrònica (dirigida per esdeveniments)
- Àmplia disponibilitat (per a majoria de SO, llenguatges, entorns de programació)
- Suporta el principals patrons d'interacció $\rightarrow$ curva d'aprenentatje ràpida
- Eficient (compromís fiabilitat/eficiència) $\rightarrow$ cues en memòria (en emissor i receptor)
- Reutilitzable. Mateix codi per a comunicar (només canviant les URL)
  - Fils en procés, processos en màquina (IPC), ordinadors en xarxa IP (TCP/IP)

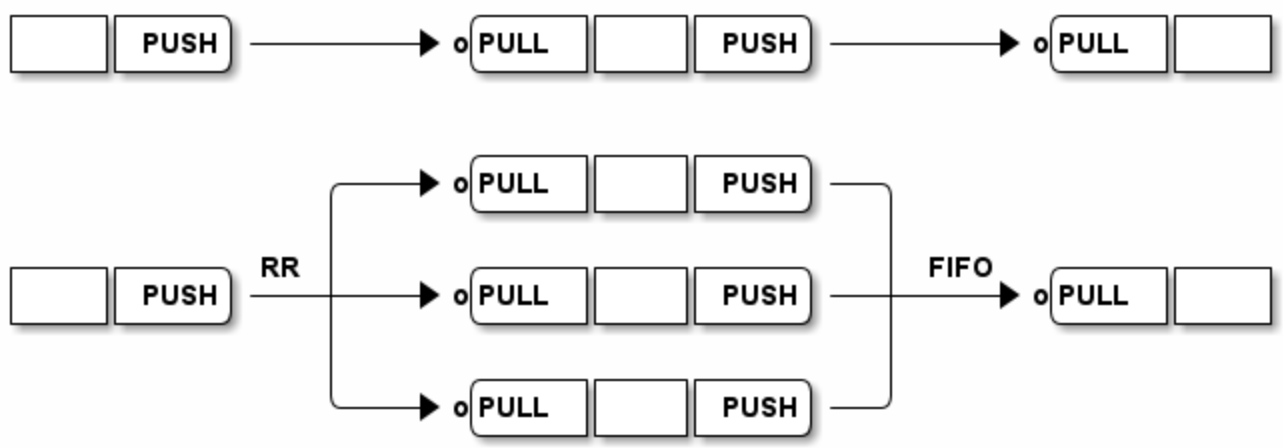
0MQ.- Tipus de Sockets i patrons de conexió

- El tipus a utilitzar dependrà dels patrons de conexió
  - cada patró té necessitats diferents (utilitza sockets diferents)
- Client/Server (sincrònic): req/rep



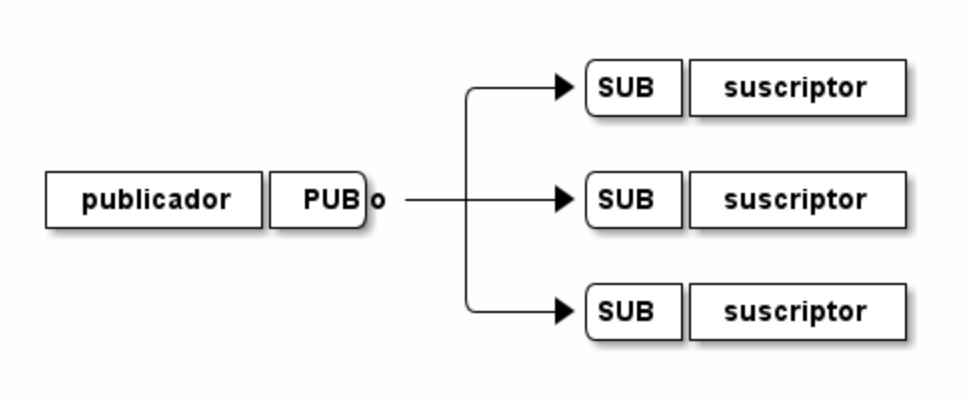
0MQ.- Tipus de Sockets i patrons de conexió

- Pipeline (unidireccional): push/pull



0MQ.- Tipus de Sockets i patrons de conexió

- Publicació/subscripció (multienviament de missatges, perque els receptors poden decidir a quins missages es subscriuen): pub/sub



0MQ.- Tipus de Sockets i patrons de conexió

- Altres patrons (ex. broker/workers): router/dealer

- 0MQ proporciona altres tipus (pair, xsub, xpub), pero NO els estudiem

0MQ.- Missatges

- Contingut dels missatges transparent per a 0MQ
  - Suporta serialització (marshalling) i reconstrucció (unmarshalling) de cadenes
  - El programador decideix com estructurar el contingut del missatge
    - Cadena (el que no siga cadena es converteix primer a cadena)
      - Les cadenes es converteixen a buffers utilitzant UTF8
      - Després les tornem a convertir a cadena
    - Podem utilitzar estandards com JSON o XML
- Els missatges es lliuren atòmicament (entrega totes les parts o no entrega res)
- Enviament i recepció asincrònics (no bloquejants)
  - Internament 0MQ gestiona el fluix de missatges entre cues del processos
- Gestió de conexió/reconexió entre agents automàtica

0MQ.- Missatges multisegment

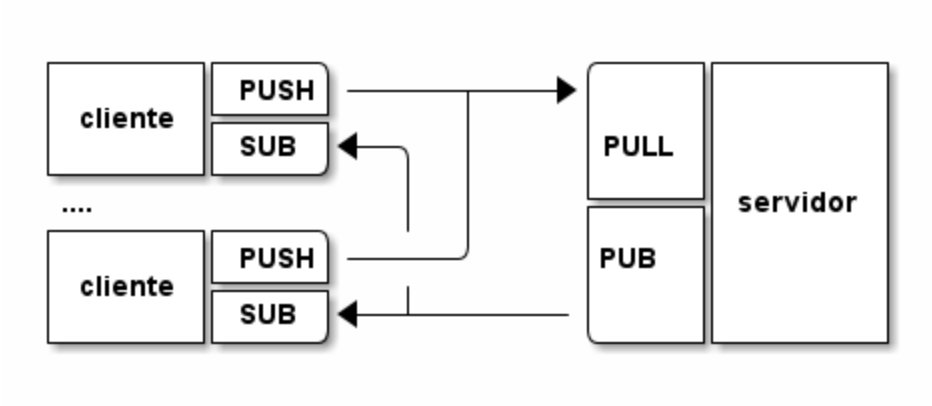
- Els missatges poden ser multisegment
  - `socket.send("text") // 4text (1 segment)`
  - `socket.send(["Hola", "", "Ana"]) // 4Hola03Ana (3 segments)`
    - En la recepció podem extraure-les automaticament (els arguments del manejador contenen els segments del missatge)

      `sock.on('message', function(s1,s2,s3) {..})`
    - O els arrepleguem en un vector

      `sock.on('message', function(...msg) {for (let seg of msg) {..}})`
  - Podem utilitzar cada segment per a una peça d'informació diferent
    - Ex. [nomAPI, versioAPI, operació, arg, ..]
  - Denominem 'delimitador' al primer segment buit ("")

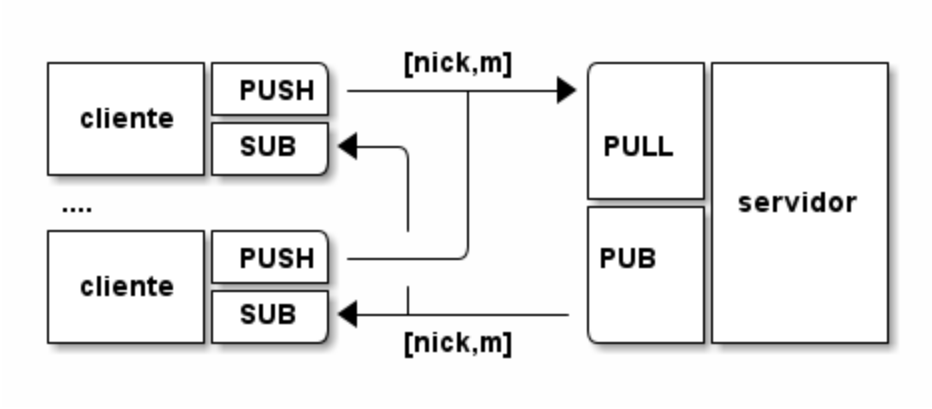
0MQ.- Passos per a desenvolupar una solució (ex. xat)

- Identifiquem els patrons d'interacció (d'on derivem tipus de sockets que necessitem, i on ubicarlos). Un xat combina:
  - Pipeline (clients push, servidor pull)
    - Cada client envia msg al servidor quan l'usuari introdueix una frase
  - Difusió (clients sub, servidor pub)
    - El servidor difón a tots els clients cada nova frase



0MQ.- Passos per a desenvolupar una solució (ex. xat)

- Defineix el format del missatges a intercamviar
  - Client a servidor: [remitent (autor de la frase = nick), text]
    - text 'HI' per a donarse d'alta, text 'BYE' per a donarse de baixa
  - servidor a clients: [autor de la frase(= nick), text]
    - nick **server** si la frase la genera el servidor (ex. per avisar alta o baixa d'un client)



0MQ.- Passos per a desenvolupar una solució (ex. xat)

- Defineix les respostes de cada agent davant els esdeveniments generats pels diferents sockets
  - Client
    - `process.stdin.on('data',(str)=>str)=>{push.send([nick,str])}) // frase escrita per teclat`
    - `process.stdin.on('end',()=>str)=>{push.send([nick,'BYE'])}) // al tancar stdin`
    - `sub.on('message', (nick,m) => {..}) // al rebre missatge del servidor`
  - Servidor
    - `pull.on('message', (nick,m) => {..}) // al rebre missatge del client`

0MQ.- Establiment connexió (bind/connect en transport TCP)

- Gestió de conexió/reconexió entre agents automàtica
- Un procés (el que hauria d'arribar primer i anar-se'n l'ultim) realitza un **bind**
  - `sock.bind('tcp://*:5555', function(err) {..})` [5555]
- Altres processos (uno o més) fan un **connect** (usant la IP i socket del que fa **bind**)
  - `sock.connect('tcp://10.0.0.1:5555', function(err) {..})`
- Quan un agent acaba executa **close** implicit. Podem invocar-ho explicitament
  - `sock.close()`
- Además de comunicació 1:1
  - N:1 $\rightarrow$ N clients (cadascun 1 connect), 1 servidor (bind)
  - 1:N $\rightarrow$ 1 client (N connects, un a cada servidor), N servidors (cadascun 1 bind)

0MQ en node

```
const zmq = require('zeromq') // importa biblioteca
let zsock = zmq.socket('tipusSocket')// creació socket (existeixen diversos tipus)
zsock.bind("tcp://*:5555") // bind en el port 5555
zsock.connect("tcp://10.0.0.1:5555") // o connect (host 10.0.0.1, port 5555)
zsock.send([..,..]) // enviament
zsock.on("message", callback) // recepció
zsock.on("close", callback) // resposta al tancament de la connexió
```

0MQ.- codi example servidor xat

```
const zmq = require('zeromq')
let pub = zmq.socket('pub')
let pull= zmq.socket('pull')
pub.bind ('tcp://*:9998')
pull.bind('tcp://*:9999')

pull.on('message', (id,txt) => {
   switch (txt.toString()) {
     case 'HI' : pub.send(['server',id+' connected']);    break
     case 'BYE': pub.send(['server',id+' disconnected']); break
     default   : pub.send([id,txt])
   }
})
```

0MQ.- codi example client xat

```
const zmq = require('zeromq')
const nick='Ana'
let sub = zmq.socket('sub')
let psh = zmq.socket('push')
sub.connect('tcp://127.0.0.1:9998')
psh.connect('tcp://127.0.0.1:9999')
sub.subscribe('') // subscrit a tots el missatges

sub.on('message', (nick,m) => {console.log('['+nick+']'+m)})

process.stdin.resume()
process.stdin.setEncoding('utf8')
process.stdin.on('data' ,(str)=> {psh.send([nick, str.slice(0,-1)])})
process.stdin.on('end',()=> {psh.send([nick, 'BYE']); sub.close(); psh.close()})
process.on('SIGINT',()=> {process.stdin.end()})

psh.send([nick,'HI'])
```