Aquest examen conté 20 qüestions d'opció múltiple. En cadascuna d'elles solament una de les seues respostes és correcta. Les contestacions han de presentar-se en una fulla lliurada a part. Les respostes correctes aporten 0.5 punts a la nota del parcial mentre que les incorrectes resten una dècima.

En la fulla de respostes has d'emplenar la casella triada acuradament. Utilitza un llapis o un bolígraf (negre o blau fosc). Es pot usar "Tipp-Ex" o algun corrector similar. En aquest cas, NO INTENTES DIBUIXAR DE NOU LA CASELLA QUE HAGES ESBORRAT.

TEORIA

1. Els sistemes distribuïts...

A	...estan formats generalment per múltiples agents que s'executen concurrentment. Aquests agents mantenen cert estat independent. Per a tenir un sistema distribuït es necessita tenir múltiples processos (o agents) que col·laboren i s'executen en ordinadors diferents facilitant una imatge de sistema únic. Malgrat aquesta col·laboració, cada procés pot mantenir algun estat local desconegut per (i inaccessible per a) els altres processos, és a dir, cert estat independent.
B	...no necessiten cap mecanisme de comunicació entre ordinadors. Aquests agents han d'executar-se en almenys dos ordinadors diferents per a tenir un sistema distribuït. Per això, aquests sistemes necessiten algun mecanisme de comunicació (normalment una xarxa) entre ordinadors.
C	...sempre utilitzen interacciones client-servidor. Hem vist al Tema 1 que hi ha altres tipus d'interacció entre agents; per exemple, interaccions *peer-to-peer*. Per tant, no totes les interaccions entre agents han d'establir-se entre clients i servidors seguint un patró petició/resposta.
D	...no tindran mai condicions de carrera. No. Les condicions de carrera poden donar-se en qualsevol sistema concurrent. Els sistemes distribuïts (per ser també concurrents) no poden garantir, per si mateixos, que les condicions de carrera no es donen mai.
E	Totes les anteriors.
F	Cap de les anteriors.

2. La computació en el núvol (*Cloud computing*)...

A	...és un dels paradigmes actuals de prestació de serveis en la computació distribuïda.

B	...té com a objectiu la prestació de serveis de còmput d'una manera escalable i eficient.
C	...segueix un model de “pagament per ús”.
D	...usa generalment infraestructures virtualitzades.
E	Totes les anteriors. La computació en el núvol està orientada a facilitar els mecanismes necessaris per a desenvolupar, desplegar i proveir serveis de programari d'una manera eficient i escalable. Normalment aquests serveis es faciliten seguint un model de “pagament per ús” i els mecanismes de virtualització s'utilitzen per a aprofitar tant com siga possible els recursos dels equips, és a dir, de la infraestructura.
F	Cap de les anteriors.

3. En un sistema distribuït, la interacció entre els seus agents...

A	…no ha de dur-se mai a terme. Si interactuaren, el sistema seria concurrent en lloc de distribuït. No. Els agents han d'interactuar. D'una altra manera, el sistema resultant seria un conjunt de processos aïllats i independents en lloc d'un sistema capaç de combinar la funcionalitat de múltiples components bàsics per a resoldre d'una manera eficient i robusta alguns problemes i tasques complexos.
B	…es realitza intercanviant missatges o compartint memòria. Sí. Aquests són els mecanismes de comunicació habituals en sistemes concurrents. Els sistemes distribuïts usen tots dos mecanismes.
C	…es realitza sense compartir memòria. La compartició de memòria està prohibida en els sistemes distribuïts. No. Quan diversos agents d'un sistema distribuït estan situats en un mateix ordinador poden compartir memòria. No hi ha cap regla que limite o prohibisca l'ús de memòria compartida en una aplicació distribuïda quan la memòria puga compartir-se entre alguns dels seus agents.
D	…s'aconsegueix quan tots els agents residisquen en un mateix ordinador. No. No té sentit obligar al fet que tots els agents d'un sistema distribuït estiguen desplegats en un únic ordinador. La fallada d'aquest ordinador comportaria la fallada completa de l’aplicació o sistema distribuït.
E	Totes les anteriors.
F	Cap de les anteriors.

4. El model de programació guarda / acció ...

A	Se segueix en la programació multi-fil, on les seccions crítiques equivalen a les guardes i els fils ("threads") equivalen a les accions. No. Una guarda és equivalent a una precondició i una secció crítica serà sempre un fragment de codi. Per tant, no poden donar-se aquestes equivalències. A més, un fil és una entitat dinàmica (comporta execució d'instruccions) i una acció podria facilitar la seqüència d'instruccions que havia d'executar el fil però, així i tot, no són conceptes equivalents (el fil és dinàmic i l'acció és estàtica ja què aquest últim concepte se sol utilitzar per a donar estructura als algorismes).
B	Se segueix en la programació asincrònica (o dirigida per esdeveniments), on els esdeveniments equivalen a les accions i les funcions "callback" dels esdeveniments equivalen a les guardes. Vegeu l'explicació de l'apartat D. Els esdeveniments són guardes i les funcions són accions.
C	Se segueix en la programació multi-fil, on els fils ("threads") equivalen a guardes que s'activen i se suspenen, i les operacions en les seccions crítiques equivalen a les accions. Vegeu l'explicació de l'apartat A. Aquests quatre conceptes no són equivalents (ni guarden una relació estreta).
D	Se segueix en la programació asincrònica (o dirigida per esdeveniments), on els esdeveniments equivalen a les guardes i les funcions "callback" dels esdeveniments equivalen a les accions. En un algorisme una acció és un bloc de sentències que té sentit per si mateix ja que defineix alguna operació útil. En el model guarda / acció, les accions estan associades a les seues precondicions (a les quals anomenem "guardes" en aquest model). Una vegada la precondició o guarda es complisca, l'acció estarà habilitada i podrà executar-se. La programació asincrònica es correspon bé amb el model guarda / acció perquè els "listeners" dels esdeveniments defineixen blocs de sentències equivalents a accions (que s'executen de manera atòmica ja que no poden ser interrompudes per altres accions habilitades) i un esdeveniment pot considerar-se equivalent a una guarda que se satisfarà cada vegada que ocórrega l'esdeveniment.
E	Totes les anteriors.
F	Cap de les anteriors.

5. Algunes característiques rellevants dels models de sistemes distribuïts:

A	Se centren en les principals propietats del comportament del sistema.
B	Faciliten una bona eina per a raonar sobre la correcció dels algorismes i protocols basats en ells.
C	El seu alt nivell d'abstracció.
D	Faciliten una base per a discutir sobre la impossibilitat de resoldre problemes en certs sistemes distribuïts (p. ex., el consens en sistemes asincrònics).
E	Totes les anteriors. Un model ha de centrar-se en les propietats essencials de l'element o sistema que represente (A) i això comporta la utilització d'un alt nivell d'abstracció (C), descartant molts detalls que es considerarien irrellevants. L'objectiu dels models és facilitar una imatge d'un sistema que permeta dissenyar algorismes capaços de resoldre problemes en el sistema real així com discutir sobre la seua correcció abans que siguen implantats en un programa (B). Aquestes discussions poden també orientar-se a avaluar la impossibilitat de resoldre alguns problemes en els sistemes modelats (D). Aquests resultats d'impossibilitat solen basar-se en les condicions que defineixen el model de sistema que s'haja suposat.
F	Cap de les anteriors.

6. Els elements a considerar en un model de sistema poden ser...

A	L'arquitectura de l'equip, el sistema operatiu, el middleware i el llenguatge de programació. Aquests elements solen ser irrellevants en dissenyar un algorisme. Els models de sistema facilitaran una imatge abstracta que resulte útil per a generar un algorisme que resolga alguns dels problemes d'en un sistema real. Per tant, aquesta llista no inclou els elements a considerar en un model de sistema.
B	Processos, esdeveniments, aspectes de comunicació, fallades, gestió del temps i nivell de sincronia. Aquests són els elements que defineixen un model de sistema: quin tipus de **processos** s'estan suposant, quins **esdeveniments** hem de considerar en l'algorisme que estiguem construint (interns, d'entrada, d'eixida...) ja que amb ells definirem la interfície del component que implante aquest algorisme, quins mecanismes de **comunicació** seran utilitzats pels agents, quins tipus de **fallades** podran ocórrer en el sistema i quines conseqüències podrà tenir cada classe de fallada, com gestionen els agents el transcurs del **temps** i la sincronia, quin nivell **de sincronia** se suposa en la interacció entre agents...
C	Nivell físic, nivell d'enllaç, nivell de xarxa, nivell de transport i nivell d'aplicació. No. Aquests són els nivells que defineixen l'arquitectura estàndard en un sistema de comunicacions de xarxa.
D	Sistema gestor de bases de dades, middleware i interfície d'usuari. No. Aquests són exemples dels elements que permeten implantar una arquitectura client/servidor multinivell.
E	Totes les anteriors.
F	Cap de les anteriors.

7. En la programació de sistemes distribuïts, l'ús del middleware és aconsellable perquè...

A	Introdueix múltiples transparències, ocultant detalls de baix nivell i oferint una interfície uniforme.
B	Té una implantació senzilla, i poca complexitat en els elements manejats.
C	Proporciona una operativa estandarditzada, comprensible i ben definida.
D	Facilita la interoperabilitat, la interacció amb productes de terceres parts.
E	Totes les anteriors. Aquests són quatre dels avantatges introduïts per un nivell middleware en una arquitectura de sistemes distribuïts.
F	Cap de les anteriors.

8. Els problemes que trobem en els sistemes distribuïts orientats a objectes són:

A	Tots els objectes semblen ser locals i això pot generar llargs intervals per a completar la seua invocació en cas que siguen remots. És un problema ja que el temps necessari per a realitzar una invocació a objecte serà difícil de predir en un entorn com aquest.
B	Els objectes mantenen estat i l'estat es compartirà entre els agents que invoquen els seus mètodes. Això pot provocar problemes de consistència. L'estat compartit pot provocar condicions de carrera i les condicions de carrera generaran inconsistències d'estat. A més, si l'objecte s'arribara a replicar els seus mètodes serien invocats per diversos agents i aquests agents podrien ser atesos per rèpliques diferents. Les modificacions generades en aquestes crides podrien, de nou, generar inconsistències entre les rèpliques.
C	El seu estat compartit necessita mecanismes de control de concurrència. Això pot ocasionar bloquejos, evitant que els sistemes siguen escalables. L'estat compartit defineix seccions crítiques. Aquestes seccions crítiques han de ser protegides mitjançant protocols d'entrada i protocols d'eixida. Aquests protocols solen implantar-se utilitzant mecanismes de control de concurrència (p. ex., locks, semàfors, monitors...) i aquests mecanismes bloquegen, quan és necessari, als agents en execució. Per tant, això pot comportar problemes greus quan desitgem desenvolupar serveis escalables.
D	Els seus mecanismes d'invocació faciliten una alta transparència d'ubicació. Això exigeix protocols de recuperació complexos per a gestionar les fallades. Quan un servidor falle serà reemplaçat per alguna de les seues rèpliques, mantenint la transparència d'ubicació. Això exigeix que el protocol de recuperació haja de reaccionar ràpidament. Els protocols que requerisquen una reacció i recuperació ràpides no són senzills i depenen del model de replicació utilitzat (actiu, passiu o alguna variant intermèdia).
E	Totes les anteriors.
F	Cap de les anteriors.

SEMINARIOS

9. Considere's el següent programa (incomplet) escrit en Node:

```
function logaritme(x,b) { return Math.log(x)/Math.log(b) }
function logBase ... // a completar
log2 = logBase(2);
log8 = logBase(8);
console.log("Logarithm base 2 of 1024 = " + log2(1024));
console.log("Logarithm base 8 of 4096 = " + log8(4096));
```

Quina implementació de la funció `logBase` seria correcta?

A	`function logBase(b) { return logaritme(x,b) }`

B	`function logBase(b) {`
	`  return function(x) { return logaritme(x,b) }`
	`}`
	logBase() ha de ser una funció que retorne una altra funció com el seu resultat (ja que log2() i log8() són funcions en aquest exemple). A més, la seua funció retornada ha de recordar l'argument utilitzat en la crida a logBase() per a emprar-ho com la base del logaritme a calcular. Addicionalment, el seu paràmetre ha de ser un nombre ("x") del que es demana el logaritme. Aquest apartat B compleix amb tots aquests requisits.
	
	L'apartat A no retorna una funció sinó un valor.
	
	L'apartat C retorna una funció que no retorna res.
	
	L'apartat D retorna una funció que realitza alguns càlculs però en la qual s'han interpretat incorrectament els paràmetres de logBase(), utilitzant-los a l'inrevés del que es devia.

C	`function logBase(x) {`
	`  return function(b) { logaritme(x,b) }`
	`}`

D	`function logBase(x) {`
	`  return function(b) { return logaritme(x,b) }`
	`}`

E	Totes les anteriors.
F	Cap de les anteriors.

10. Considere's el següent programa escrit en Node:

```
var fruits = ["Banana", "Orange", "Lemon", "Apple"];
var numbers = [7, 3, "Cloud", 9];
var funcs = [function(x) {return 2*numbers[x]},
             function(x) {return fruits[x]}];
var s = "";
for (var i=0; i<2; i++)
  for (var j=0; j<5; j=j+2)
     s += funcs[i](j) + ", ";
console.log(s);
```

En executar-ho, l'eixida que es mostrarà en consola serà:

A	14, 6, NaN, Banana, Orange, Lemon,
B	14, NaN, NaN, Banana, Lemon, undefined, Aquest programa conté dos bucles niats. L'extern (amb la variable "i") realitza dues iteracions, amb els valors 0 i 1 per a "i". L'intern (amb la variable "j") realitza tres iteracions, amb els valors 0, 2 i 4 per a "j". En cada iteració la seua única instrucció concatena a la cadena "s" (inicialment buida) el resultat de cridar a la funció "funcs[i]" passant com a argument el valor de "j". La funció funcs[0] retorna com a resultat el doble del valor contingut en numbers[j]. Aquest valor ha de ser un nombre (d'una altra manera, el resultat de l'operador "*" és NaN). La funció funcs[1] retorna la component "j" en el vector fruits[]. Per tant, mentre i valga 0, els valors retornats en la crida a funcs[0] seran: el doble de 7 (14), el doble de "Cloud" (NaN) i el doble de "undefined" (NaN). Quan valga 1 els valors retornats seran "Banana" (és a dir, fruits[0]), "Lemon" (fruits[2]) i undefined (fruits[4]). Amb això la seqüència obtinguda en aquesta execució és: "14, NaN, NaN, Banana, Lemon, undefined,"
C	14, 2Cloud, undefined, Banana, Lemon, undefined,
D	Banana, 14, Lemon, NaN, undefined,
E	No es mostraria res, excepte un missatge d'error indicant que l'array `numbers` està mal definit, per contenir valors de diferents tipus.
F	Cap de les anteriors.

NOTA: En el tercer element de la llista donada com a eixida, el valor presentat és NaN ja que s'està multiplicant 2 per *undefined* en accedir a una component inexistent d'un vector (numbers[4]). No obstant això, també es podria pensar que el resultat d'aquesta multiplicació és *undefined*, per la qual cosa s'admetrà F (cap de les anteriors) com a resposta vàlida encara que no ho siga.

11. Considere's la següent funció escrita en Node:

```
function f(x,y) {
   x = x || 'taronja'; y = y || 98;
   console.log('x='+x+' y='+y);
}
```

Indique quina seria l'eixida que es mostrarà en consola si s'executa:

```
f(36);          f(undefined,'poma');    f(45,0,67);
```

A	x=36 y=98	x=undefined y=poma	x=45 y=0
B	x=36 y=98	x=taronja y=poma	x=45 y=0
C	x=36 y=98	x=taronja y=poma	x=45 y=98
	Aquesta qüestió versa sobre l'operador lògic '||' (OR) i la utilització d'arguments en les crides a funció. L'operador '||' retorna el seu operand esquerre quan aquest no puga considerar-se fals (és a dir, la constant Booleana **false**) i retornarà el seu operand dret en un altre cas. No obstant això, en Javascript hi ha diversos valors falsos. Quan considerem nombres, 0 és **false** i tots els altres valors són **true**. D'altra banda, *undefined* també és **false**. Per tant, en la crida f(36) estem utilitzant solament un argument per a f i f té dos paràmetres. Això implica que el paràmetre "y" rebrà un valor *undefined*. A causa d'això, "y" prendrà en aquest cas el valor 98. Així, en aquesta primera crida tindrem **x=36** i **y=98**. En la crida f(undefined, 'poma'), la variable "x" obtindrà el valor **taronja** ja que aquest últim és l'operand dret en la instrucció "x = x || 'taronja'". Per tant, tindrem **x=taronja** i **y=poma**. Finalment, en la crida f(45,0,67), l'argument 67 (el tercer) serà descartat i la "y" obtindrà el valor 98 perquè 0 és equivalent a **false**. Amb això, **x=45** i **y=98**. L'apartat C és l'únic que presenta valors correctes per a aquestes tres crides.		
D	x=36 y=36	x=taronja y=poma	x=45 y=67
E	No es mostraria res, excepte missatges d'error ja que hi ha invocacions incorrectes (pel seu nombre d'arguments) de la funció f.		
F	Cap de les anteriors.		

12. Considere's el següent programa escrit en Node:

```
var eve = new (require('events')).EventEmitter;
var s = "print";
var n = 0;
var handler = setInterval( function(){eve.emit(s);}, 1000 );
eve.on(s, function() {
   if ( n < 2 ) console.log("Event", s, ++n, "times.");
   else clearInterval(handler);
});
```

Si s'executa aquest programa indique, en relació a l'eixida que es mostrarà en consola i al temps d'execució, quina de les següents opcions és la correcta:

<table>
<tr><td>A</td><td>Event print 1 times.
Event print 2 times.</td><td>I conclouria després de 3 segons.</td></tr>
<tr><td colspan="3">Aquest programa genera un esdeveniment “print” cada segon (línia 4). En el listener per a “print” es comprova si n és menor que 2 (inicialment és zero) i, en aquest cas, s'imprimeix un missatge (començant amb n=1 ja que aquesta variable es preincrementa en els arguments de la instrucció console.log). En un altre cas s'elimina l'interval de generació de l'esdeveniment “print”. Quan això ocórrega, el procés finalitzarà perquè no hi ha més esdeveniments que manejar ni altres torns pendents.
Amb això, quan s'inicia aquest procés, la seua línia 4 programa la generació de l'esdeveniment 4 cada segon.
Un segon després s'imprimeix el missatge “Event print 1 times”.
De nou, un segon després s'imprimeix “Event print 2 times”.
En el tercer segon, el listener comprova si n és menor que 2 però ara ja és 2. Per tant, s'executa l'else i es cancel·len els intervals. Com a resultat d'això, el procés finalitza tres segons després del seu inici havent-hi imprés dos missatges.</td></tr>
<tr><td>B</td><td>Event print 1 times.
Event print 2 times.
Event print 3 times.</td><td>I conclouria després de 4 segons.</td></tr>
<tr><td>C</td><td>Event print 1 times.
Event print 2 times.
…</td><td>I no conclouria. Cada segon, mostraria una nova línia amb el nombre incrementat en una unitat.</td></tr>
<tr><td>D</td><td>Event print 0 times.
Event print 1 times.
…</td><td>I no conclouria. Cada 10 segons, mostraria una nova línia amb el nombre incrementat en una unitat.</td></tr>
<tr><td>E</td><td>No es mostraria res, perquè no està ben definit l'objecte listener.</td><td>I no conclouria, perquè s'emet cíclicament l'esdeveniment “print”.</td></tr>
<tr><td>F</td><td colspan="2">Cap de les anteriors.</td></tr>
</table>

13. Considerant el programa següent...

```
var http = require('http');
var fs = require('fs');
http.createServer(function(request,response) {
  fs.readdir(__dirname, function(err,data) {
    if (err) {
      response.writeHead(404, {'Content-Type':'text/plain'});
      response.end('Unable to read directory '+__dirname);
    } else {
      response.writeHead(200, {'Content-Type':'text/plain'});
      response.write('Directory: ' + __dirname + '\n');
      response.end(data.toString());
    }
  })
}).listen('1337');
```

Seleccione les opcions correctes:

A	Aquest programa genera una excepció i avorta en cas de no poder llegir el contingut del directori actual. No. Si trobara un error tractant de llegir el directori, el *callback* per a la crida a readdir() rebria un objecte en el seu primer argument i el servidor web retornaria una resposta a la sol·licitud del client, sense generar cap excepció ni avortar.
B	Aquest programa és un servidor web que respon amb el nom i llista de fitxers en el directori actual. Aquesta és la descripció correcta de les tasques desenvolupades en aquest programa. L'operació http.createServer() facilita la base per a escriure un servidor HTTP. Independentment de quines peticions s'hagen rebut, aquest procés sempre contesta amb una resposta que conté el nom i la llista de fitxers continguts en el directori on haja sigut iniciat. El nom d'aquest directori es manté en la variable __dirname.
C	Aquest programa no funciona perquè no ha declarat la variable "__dirname" i no ha importat el mòdul 'process' on està definida. No. "__dirname" és una variable que ja està declarada per omissió. No es necessita importar cap mòdul ni declarar-la per a poder-la usar.
D	Aquest programa no funciona perquè 'data' és un vector de noms de fitxer i els vectors no poden ser transformats en cadenes. Fals. Un vector de cadenes (perquè els noms de fitxer són cadenes) pot ser convertit en cadena sense major problema.
E	Totes les anteriors.
F	Cap de les anteriors.

14. Alguns problemes de l'algorisme d'exclusión mútua amb servidor central:

A	No compleix la seua condició de vivacitat. Tots els algorismes distribuïts correctes han de complir les seues condicions de vivacitat i seguretat. Aquest algorisme respecta la seua condició de vivacitat, és a dir, assegura que tots els sol·licitants aconseguiran accedir a la secció crítica en algun moment.
B	No compleix la seua condició de seguretat. Tots els algorismes distribuïts correctes han de complir les seues condicions de vivacitat i seguretat. Aquest algorisme respecta la seua condició de seguretat, és a dir, assegura que no hi haurà mai més d'un procés simultàniament en la secció crítica.
C	Necessita més missatges que els altres algorismes vistos en el Seminari 2 per a resoldre el problema d'exclusió mútua. No. De fet és un dels algorismes que necessita menys missatges per a gestionar una secció crítica.
D	És fràgil en situacions de fallada. El servidor central és un punt únic de fallada. Cert. Si el servidor central fallara cap dels participants podria superar el seu protocol d'entrada a la secció crítica perquè quedarien esperant el missatge d'autorització d'entrada que hauria d'enviar-los el servidor central.
E	Totes les anteriors.
F	Cap de les anteriors.

15. Els algorismes d'elecció de líder...

A	...són un subconjunt dels algorismes de consens. Sí. Per a triar un líder els processos han d'aconseguir un consens sobre quin és el millor candidat.
B	...necessiten que tots els processos tinguen un identificador diferent. Sí. D'una altra manera seria impossible triar a algun d'ells ja que aquesta decisió està basada en la comparació dels seus identificadors.
C	...usen un criteri determinista per a triar al líder. Sí, i el criteri ha de ser conegut per tots els processos participants.
D	...exigeixen que es trie solament a un procés. Sí. El líder ha de ser únic.
E	Totes les anteriors.
F	Cap de les anteriors.

16. Suposem que es necessita implantar un servei de xat utilitzant node.js i ØMQ. El servidor difon els missatges dels usuaris i no ha de suspendre's mai tractant d'enviar un missatge (de qualsevol tipus). Els programes clients envien els missatges dels usuaris al servidor, esperen els missatges reexpedits pel servidor i informen al servidor quan un usuari s'incorpora o abandona el sistema. Per a implantar aquest servei de xat...

A	El servidor ha d'usar un socket PULL i un altre REP per a interactuar amb els clients. No. Els missatges dels usuaris han de ser difosos. Solament els sockets PUB poden difondre un missatge utilitzant una única crida a send().

B	El servidor ha d'usar un socket SUB i un altre REQ per a interactuar amb els clients. No. Els missatges dels usuaris han de ser difosos. Solament els sockets PUB poden difondre un missatge utilitzant una única crida a send().
C	El servidor ha d'usar un socket PUB i un altre PULL per a interactuar amb els clients. Sí. Els missatges dels usuaris han de ser difosos pel servidor i solament un socket PUB pot realitzar aquesta difusió amb una única crida a send(). A més, es necessitarà un altre socket per a acceptar i processar els missatges enviats pels processos clients per a dir-li al servidor que un usuari s'ha incorporat o ha abandonat l'aplicació. Aquest segon socket ha d'admetre recepcions de missatges i un socket PULL pot gestionar això d'una manera asincrònica, sense bloquejar mai el servidor en el seu processament dels missatges.
D	El servidor ha d'usar un socket REP i un altre SUB per a interactuar amb els clients. No. Els missatges dels usuaris han de ser difosos. Solament els sockets PUB poden difondre un missatge utilitzant una única crida a send().
E	Totes les anteriors.
F	Cap de les anteriors.

17. Suposem que hem implantat un servei suportat per múltiples (p. ex., 10) processos servidors situats en ordinadors diferents. Aquests servidors utilitzen sockets REP i els seus clients usen sockets REQ. Si construïm un broker amb un socket ROUTER com a *front-end* i un socket DEALER com a *back-end* (i per a tots dos es realitza un bind()), llavors...

A	Els clients no necessiten conèixer quants processos servidors hi ha. Sí. El broker és l'únic procés que interactua directament amb els servidors. Per tant, els processos clients no necessiten cap informació sobre els processos servidors.

B	Els clients no necessiten conèixer les adreces i ports de cada procés servidor. Sí. El broker és l'únic procés que interactua directament amb els servidors. Per tant, els processos clients no necessiten cap informació sobre els processos servidors.
C	La quantitat de processos servidors pot variar dinàmicament. Ells han de connectar-se al socket *back-end* perquè el broker puga utilitzar-los. Sí. Podem modificar la quantitat de servidors d'una manera transparent. Solament necessiten connectar-se al socket DEALER.
D	El broker no ha de modificar cap segment dels missatges per a propagar-los del *front-end* al *back-end* i del *back-end* al *front-end*. Sí. Tots dos sockets (ROUTER i DEALER) no necessiten preocupar-se pel contingut dels missatges. Cap dels segments en els missatges necessita ser modificat (ni afegit ni eliminat). Amb aquesta estratègia els sockets DEALER distribueixen de manera circular els missatges de petició entre tots els servidors connectats.
E	Totes les anteriors.
F	Cap de les anteriors.

Per a contestar a las següents 2 qüestions (nº 18 i 19), consideren-se els següents programes Node amb ØMQ. Un servidor (*server.js*):

```
var zmq = require('zmq')
var rep = zmq.socket('rep')
rep.bindSync('tcp://127.0.0.1:'+process.argv[2])
var n = 0
rep.on('message', function(msg) {
  console.log('Request: ' + msg)
  rep.send('World ' + ++n)
})
```

I un client (*client.js*):

```
var zmq = require('zmq')
var req = zmq.socket('req')
req.connect('tcp://127.0.0.1:'+process.argv[2])
req.connect('tcp://127.0.0.1:'+process.argv[3])
var n = 0
setInterval( function() { req.send('Hello ' + ++n) }, 100 )
req.on('message', function(msg) {
  console.log('Response: ' + msg)
})
```

18. Consideren-se els anteriors programes Node amb ØMQ (*server.js* i *client.js*). Si, en 3 terminals, s'executaren 2 servidors i 1 client mitjançant:

```
node server 8001
node server 8002
node client 8001 8002
```

Les primeres línies que es mostraran en les terminals dels servidors seran:

<table>
<tr><td>A</td><td>En una terminal:
<code>Request: Hello 1</code>
<code>Request: Hello 3</code>
I en l'altra terminal:
<code>Request: Hello 2</code>
<code>Request: Hello 4</code>
Ja que el socket REQ del client està connectat als REP dels servidors, aquest REQ distribueix de manera circular els seus missatges entre ells. Això implica que el seu primer missatge anirà al primer servidor, la segona petició al segon servidor, la tercera petició al primer servidor i així successivament.
Per això, els missatges s'estaran imprimint tal com es mostra en aquest apartat ja que cada petició incrementa el mateix comptador local (n) i la primera petició va incloure el valor 1 en el seu missatge.</td></tr>
<tr><td>B</td><td>En ambdues terminals:
<code>Request: Hello 1</code>
<code>Request: Hello 2</code>
No. Veure l'explicació en l'apartat A.</td></tr>
<tr><td>C</td><td>En ambdues terminals (sent x, y, z… nombres tals que x < y < z < …):
<code>Request: Hello x</code>
<code>Request: Hello y</code>
<code>Request: Hello z</code>
<code>...</code>
No. Veure l'explicació en l'apartat A.</td></tr>
<tr><td>D</td><td>En una terminal:
<code>Request: Hello 1</code>
<code>Request: Hello 2</code>
I en l'altra terminal:
<code>Request: Hello 3</code>
<code>Request: Hello 4</code>
No. Veure l'explicació en l'apartat A.</td></tr>
<tr><td>E</td><td>No es mostraria res, atès que el client no sabria a quin dels servidors enviar les seues peticions. (Per a un funcionament correcte, el client hauria de connectar-se a un socket ROUTER).
No. Veure l'explicació en l'apartat A.</td></tr>
<tr><td>F</td><td>Cap de les anteriors.</td></tr>
</table>

19. Consideren-se els mateixos programes, i el mateix escenari d'execució, de la qüestió anterior. Les primeres línies que es mostraran en la terminal del client seran:

A	`Response: World 1`
	`Response: World 2`
	`Response: World 3`
	`Response: World 4`
	No. Veure l'explicació en l'apartat B.

B	`Response: World 1`
	`Response: World 1`
	`Response: World 2`
	`Response: World 2`
	Sí. Tal com s'ha explicat en la qüestió 18, el client envia cada petició a un servidor diferent, equilibrant la càrrega de tots dos. Cada servidor utilitza un comptador local per a etiquetar la seua resposta. Per tant, la primera resposta rebuda pel client és la primera enviada pel primer servidor, la segona és la primera resposta del segon servidor, la tercera és la segona resposta del primer servidor i així successivament.

C	`Response: World 1`
	`Response: World 3`
	`Response: World 5`
	`Response: World 7`
	No. Veure l'explicació en l'apartat B.

D	`Response: World 1`
	`Response: World 3`
	`Response: World 2`
	`Response: World 4`
	No. Veure l'explicació en l'apartat B.

E	No es mostraria gens, atès que, com el client no sabria a quin dels servidors enviar les seues peticions, cap dels servidors podria enviar respostes. No. Veure l'explicació en l'apartat B.
F	Cap de les anteriors.

20. Consideren-se els següents programes Node amb ØMQ. Un publicador:

```
var zmq = require('zmq')
var pub = zmq.socket('pub').bindSync('tcp://*:5555')
var count = 0
setInterval(function() {
  pub.send('PRG ' + count++)
  pub.send('TSR ' + count++)
}, 1000)
```

I un subscriptor:

```
var zmq = require('zmq')
var sub = zmq.socket('sub')
sub.connect('tcp://localhost:5555')
sub.subscriu('TSR')
sub.on('message', function(msg) {
  console.log('Received: ' + msg)
})
```

Si s'executara, en primer lloc, el publicador i, tres segons després, el subscriptor, les primeres línies de l'eixida que es mostraran en la terminal del

subscriptor seran:

A	`Received: TSR 1`
	`Received: TSR 2`
	`Received: TSR 3`
	…
	No. Veure l'explicació en l'apartat D.

B	`Received: TSR 1`
	`Received: TSR 3`
	`Received: TSR 5`
	…
	No. Veure l'explicació en l'apartat D.

C	`Received: PRG 4`
	`Received: TSR 5`
	`Received: PRG 6`
	…
	No. Veure l'explicació en l'apartat D.

D	`Received: TSR 5`
	`Received: TSR 7`
	`Received: TSR 9`
	…
	Sí. Ha de considerar-se que el publicador envia dos missatges per segon, cadascun amb un prefix diferent, però utilitzant el mateix comptador que incrementem en cada enviament. Per tant, els missatges enviats pel publicador són...
	En el segon 1: PRG 0, TSR 1
	En el segon 2: PRG 2, TSR 3
	En el segon 3: PRG 4, TSR 5
	En el segon 4: PRG 6, TSR 7
	
	El subscriptor solament rebrà els missatges amb el prefix TSR. És iniciat en el segon 3. Com els canals PUB-SUB no tenen una persistència forta, aquells missatges difosos abans que el subscriptor es connecte s'hauran perdut. Per això, el subscriptor rebrà tots els missatges TSR a partir del 5; és a dir, TSR 5, TSR 7, TSR 9...

E	`Received: PRG 0`
	`Received: TSR 1`
	`Received: PRG 2`
	…
	No. Veure l'explicació en l'apartat D.

F	Cap de les anteriors.