


A


Aquest examen conté 20 qüestions d'opció múltiple. En cada pregunta només una de les seues respostes és correcta. Les contestacions han de presentar-se en una fulla entregada a part. Les respostes correctes aporten 0.5 punts mentre que las incorrectes resten 0.167 punts.

TEORIA

1. El model de servei PaaS per a la computació en el núvol...

a	...té com a principal objectiu la virtualització del <i>maquinari</i> . Fals. La virtualització del <i>maquinari</i> és un aspecte que ha de ser gestionat en el nivell d'infraestructura. El model de “plataforma com a servei” (PaaS) se situa sobre el nivell de gestió d'infraestructura (que seria el model IaaS en un entorn de computació en el núvol).
b	...proporciona un servei d'aplicacions distribuïdes als seus usuaris directes. Fals. El model de “programari com a servei” (SaaS) és el responsable d'aquesta classe de serveis. El model PaaS se situa per baix del SaaS en una arquitectura de serveis estructurada en nivells.
	...gestiona la infraestructura subjacent i automatitza el desplegament i l'administració d'aplicacions per als proveïdors d'aquestes. Vertader. Aquestes són les tasques a considerar en el model PaaS.
d	...evita en la mesura que siga possible la concurrència. Fals. Els models de servei per a la computació en el núvol no han d'evitar la concurrència. Tots els sistemes distribuïts són exemples de sistemes concurrents. La concurrència forma part de qualsevol servei distribuït. Per tant, els models de servei han de ser concurrents.

2. La Vikipèdia...

a	...és un exemple d'aplicació distribuïda que utilitza la interacció “peer-to-peer”. Per tant, és un servei fàcilment escalable. Fals. La Vikipèdia utilitza interaccions client/servidor.
b	...utilitza els navegadors web com un tipus específic d'agent servidor. Fals. Els navegadors web són exemples d'agents clients.
c	...emmagatzema la seua informació persistent en els nodes clients. Fals. La seua informació persistent es manté en bases de dades en el domini servidor, no en els nodes clients.
	...utilitza <i>proxies</i> inversos per a millorar el seu rendiment. Vertader. Aquesta és una de les tècniques explicades en el Tema 1 per a millorar el rendiment i l'escalabilitat. Els <i>proxies</i> inversos han sigut utilitzats en la Vikipèdia des de les primeres edicions d'aquest sistema.

3. LAMP són les sigles de:

a	<i>Light and Available Multi-Processing</i> (és a dir, multiprocessament lleuger i disponible). Fals.
----------	--

A

b	Linux, Apache, MySQL i PHP. Vertader. Els sistemes LAMP es van explicar en el Tema 1 (en la secció dedicada a la Vikipèdia) per a descriure com són les arquitectures habituals dels servidors web.
c	Linux, Acrobat, MongoDB i Python. Fals.
d	Linux, Apache, Memcache i PostgreSQL. Fals.

4. Un model de sistema...

a	...estableix l'arquitectura del sistema distribuït. Fals. L'arquitectura d'un sistema o servei distribuït no s'arriba a definir en el seu model de sistema. Els models de sistema no consideren aquests aspectes.
b	...descriu acuradament els equips i protocols de comunicació que han d'usar-se en un sistema distribuït real. Fals. Els equips de comunicació no es descriuen en un model de sistema.
c	...facilita una imatge d'alt nivell d'un sistema, especificant les propietats que han de considerar-se en escriure algorismes en el sistema. Vertader. Aquest seria un resum vàlid del que ha de considerar-se en un model de sistema per a un sistema distribuït, segons el que es va explicar al Tema 2.
d	...especifica tots els detalls i algorismes que estan sent utilitzats en <i>un middleware</i> . Fals. Un model de sistema no proporciona cap detall sobre parts del programari. Només les propietats principals arriben a considerar-se i això es fa a un alt nivell d'abstracció. Aquestes propietats han de ser rellevants a l'hora d'escriure els algorismes que proporcionen alguna solució als problemes existents en l'entorn.

5. Sobre el temps i la sincronització en sistemes distribuïts:

a	Un model de sistema convé que siga sincrònic ja que la sincronia evita les condicions de carrera. Fals. No tots els tipus de sincronia guarden relació amb les condicions de carrera. A més, quanta més sincronització s'assumeixca en un model de sistema, major dificultat hi haurà per a facilitar aquest comportament assumit en el model de sistema.
b	La sincronització ha d'evitar-se tant com siga possible ja que compromet el rendiment i l'escalabilitat. Vertader. Els mecanismes de sincronització bloquejaran l'activitat dels agents en alguns casos. Per això, no són convenients per a aconseguir nivells alts de rendiment i escalabilitat.
c	La comunicació sincrònica no bloqueja mai els processos. Fals. La comunicació sincrònica bloqueja els processos emissors mentre no reben algun tipus de resposta per part dels receptors.
d	Els rellotges locals de cada node poden sincronitzar-se sense intercanviar missatges. Per a fer això s'utilitza l'algorisme de Cristian. Fals. L'algorisme de Cristian està basat en l'intercanvi de missatges. Tots els algorismes de sincronització de rellotges necessiten que cada procés interactue amb algun agent extern.

A

6. En el paradigma de programació asincrònica...

a	<p>Hi ha múltiples fils en cada procés.</p> <p>Fals. No és necessari tenir múltiples fils en cada procés en aquest model de programació. Node.js és un exemple on no hi ha múltiples fils per procés.</p>
b	<p>Tots els esdeveniments ocorren simultàniament.</p> <p>Fals. Cada esdeveniment pot ocórrer en un instant diferent.</p>
c	<p>Si múltiples esdeveniments ocorren alhora, llavors les seues accions estaran habilitades però s'executaran seqüencialment i amb garanties d'atomicitat.</p> <p>Vertader. Totes les accions associades amb esdeveniments són atòmiques (és a dir, s'executen en un sol torn i no poden ser interrompudes) i, a causa d'això, seran executades una després d'una altra (és a dir, seqüencialment).</p>
d	<p>Els programes no poden estar basats en <i>callbacks</i>.</p> <p>Fals. Poden estar basats en <i>callbacks</i> i, de fet, en aquesta assignatura hem utilitzat un llenguatge de programació (JavaScript) que utilitza aquesta aproximació.</p>

7. Tots els sistemes (*middleware*) de missatgeria...

a	<p>...són exemples de servei d'autenticació (és a dir, un servei relacionat amb la seguretat).</p> <p>Fals. Els serveis d'autenticació no poden considerar-se, en el cas general, un exemple de sistema de missatgeria.</p>
b	<p>...no estan obligats a facilitar transparència d'ubicació.</p> <p>Vertader. La comunicació orientada a missatges té com a principal objectiu un bon rendiment. La transparència d'ubicació no és un objectiu en aquests sistemes. Hem treballat amb un exemple en TSR: ØMQ. En aquest <i>middleware</i> les operacions <code>bind()</code> i <code>connect()</code> no proporcionen transparència d'ubicació al programador ja que aquest ha d'especificar l'adreça IP i el port en cadascuna d'aquestes operacions. Per a proporcionar comunicació amb transparència d'ubicació s'haurien d'utilitzar noms (en lloc d'adreces) per a referir-se a altres agents.</p>
c	<p>...utilitzen <i>proxies</i> i esquelets per a millorar el seu rendiment.</p> <p>Fals. Els <i>proxies</i> i esquelets s'utilitzen en els mecanismes d'invocació remota de mètodes, facilitant les mateixes interfícies que els agents o objectes remots amb els quals haja d'interactuar-se. Els sistemes de missatgeria no utilitzen aquest tipus d'interfície. Normalment, usen operacions <code>send()</code> proporcionades per <i>sockets</i>.</p>
d	<p>...utilitzen algorismes centralitzats i depenen d'un gestor de comunicacions (<i>broker</i>) central.</p> <p>Fals. Els sistemes de missatgeria poden tenir un gestor de comunicacions o no tenir-ne cap. Per tant, no depenen en tots els casos d'aquest tipus de gestor. Fins i tot en el cas que s'utilitzara un gestor, aquest no necessitarà molts passos (és a dir, un algorisme que no siga trivial) per a gestionar la comunicació entre diferents agents. A més, els sistemes de missatgeria més eficients no solen tenir gestor, com hem vist en el cas de ØMQ.</p>

A

8. La comunicació no persistent basada en missatges...

a	...bloqueja al servidor mentre el receptor no haja notificat que ha pogut entregar el missatge. Fals. Aquesta no és la definició de comunicació no persistent sinó la de la comunicació sincrònica.
b	...facilita transparència d'ubicació. Fals. El grau de persistència en un sistema de comunicacions no està relacionat amb la transparència d'ubicació. Per a aconseguir transparència d'ubicació necessitem <i>stubs</i> en tots dos agents (emissor i receptor) proporcionant interfícies locals idèntiques a les de l'agent remot.
c	...garanteix transparència de fallades. Fals. El grau de persistència en un sistema de comunicacions no està relacionat amb la transparència de fallades. Per a aconseguir transparència de fallades hauríem de replicar els agents emissors i receptors (per a ocultar les fallades en els processos) i reintentar les accions d'enviament (per a superar la pèrdua de missatges).
d	...no permet que la comunicació s'iniciï fins que emissor i receptor puguin gestionar la transmissió del missatge. Vertader. La comunicació no persistent utilitza canals que no poden guardar els missatges que estan sent propagats. Per això, tots dos agents de comunicació han d'estar preparats abans d'iniciar la transmissió de qualsevol missatge.

SEMINARIS

9. Si considerem aquest programa...

```
var fs=require('fs');
if (process.argv.length<5) {
    console.error('Es necessiten més noms de fitxers!!');
    process.exit();
}
files = process.argv.slice(2);
files.forEach( function (name) {
    fs.readFile(name, 'utf-8', function(err,data) {
        if (err) {
            console.log(err);
        } else
            console.log('Fitxer '+name+' : '+data.length+' bytes. ');
    })
})
console.log("S'han processat "+files.length+" fitxers.");
```

Selecció de l'opció correcta, assumint que no hi haurà errors en la seua execució:

a	El programa necessita rebre almenys 5 noms de fitxer com a arguments des de la línia d'ordres. Fals. Necessita almenys 5 arguments però el primer d'ells és sempre "node" (el nom de l'interpret) i el segon el nom del programa a executar. Per tant, solament s'està sol·licitant que hi haja 3 noms de fitxer com a mínim (en lloc de 5, que és el que es diu en l'enunciat).
----------	---

A

b	El programa mostra el nom i la grandària de cadascun dels fitxers rebuts com a arguments. Vertader. Aquesta és la funcionalitat facilitada per aquest programa.
c	L'última línia que mostra és "S'han processat N fitxers." , sent N el nombre de fitxers rebuts com a arguments. Fals. La sentència que mostra aquest missatge està al final del programa però JavaScript en un llenguatge de programació asincrònic. Per això, aquesta sentència arriba a executar-se en el primer torn mentre que els altres console.log() (situats al <i>callback</i> de la funció readFile()) s'executaran en torns posteriors. Això implica que el missatge esmentat en aquest enunciat s'imprimirà en començar l'execució del programa (és a dir, serà la primera cosa que es mostre en pantalla, no l'última).
d	El programa pot ser executat en un sol torn. Fals. Necessitarà múltiples torns, com s'ha descrit en l'apartat "c".

10. Considerant el programa vist en la pregunta anterior, seleccione l'opció correcta:

a	No podrà escriure el nom de fitxer apropiat en cada iteració. En el seu lloc, sempre imprimeix el nom de l'últim fitxer. Fals. El nom de fitxer s'ha gestionat correctament en <i>el callback</i> de la funció readFile() ja que aquesta funció està, al seu torn, en el <i>callback</i> del bucle forEach() i el nom de fitxer s'està rebent com a argument en aquesta funció més externa. Així, l'esquema resultant és similar a una clausura i garanteix que cada invocació del <i>callback</i> de readFile() es faça en un àmbit que gestiona el valor apropiat per a la variable "name".
b	Genera una excepció i avorta si ocorre algun error quan tracta de llegir algun fitxer. Fals. Si es donara algun error en aquest punt, es comunicaria mitjançant el paràmetre "error" del <i>callback</i> de readFile(). En aquest cas es mostrarà un missatge d'error i no es generarà cap excepció.
c	En cas d'error, mostra un missatge en la pantalla i continua. Vertader. Ja s'ha explicat en la justificació de l'apartat anterior.
d	Mostra la mateixa grandària de fitxer en totes les iteracions. Necessitaria una clausura per a corregir aquest defecte. Fals. Com ja s'ha explicat en l'apartat "a", ja s'està utilitzant una clausura en aquesta solució.

11. Algunes condicions de seguretat per a tots els algorismes d'exclusió mútua vistos en el Seminari 2 són...

a	Un procés accedeix a la secció crítica després de completar el seu protocol d'eixida. Fals. Un procés completa el seu protocol d'eixida quan ha abandonat la seua secció crítica. Per tant, quan això succeeixca estarà en la seua secció restant (és a dir, fora de la secció crítica).
----------	---

A

b	Els processos han d'utilitzar canals de comunicació asincrònics. Fals. Cap dels algorismes presentats en el seminari necessitava canals de comunicació asincrònics.
c	No pot haver-hi més d'un procés executant la secció crítica. Vertader. Aquesta és la condició de correcció aplicable a totes les solucions per a aquest problema. Les condicions de seguretat han d'establir les restriccions que han de ser respectades per les solucions correctes. És a dir, les condicions de seguretat garanteixen que no ocorregi res incorrecte o invàlid mentre s'estiguen executant les tasques descrites en un algorisme.
d	No inanició: ha de limitar-se el nombre de vegades que un procés en el seu protocol d'entrada cedisca el pas a altres processos. Fals. Aquesta és una condició de vivacitat. No és una condició de seguretat.

12. Si considerem aquest programa...

```
var ev = require('events');
var emitter = new ev.EventEmitter;
var num1 = 0;
var num2 = 0;
function emit_i1() { emitter.emit("i1") }
function emit_i2() { emitter.emit("i2") }
emitter.on("i1", function() {
  console.log( "L'esdeveniment i1 ha ocorregut " + ++num1 + "
vegades." ));
emitter.on("i2", function() {
  console.log( "L'esdeveniment i2 ha ocorregut " + ++num2 + "
vegades." ));
emitter.on("i1", function() {
  setTimeout( emit_i2, 3000 )});
emitter.on("i2", function() {
  setTimeout( emit_i1, 2000 )});
setTimeout( emit_i1, 2000 );
```

Selecione l'afirmació correcta:

a	L'esdeveniment "i1" ocorre només una vegada, dos segons després de que el programa siga iniciat. Fals. L'esdeveniment "i1" ocorre per primera vegada dos segons després d'haver-se iniciat el programa. Posteriorment, ocorre cada cinc segons. Per tant, no ocorre només una vegada.
b	L'esdeveniment "i2" no ocorre mai. Fals. L'esdeveniment "i2" ocorre tres segons després de cada ocurrència de l'esdeveniment "i1". Per tant, ocorrerà (aproximadament) tantes vegades com ocorregui "i1".
c	L'esdeveniment "i2" ocorre cada cinc segons. Vertader. Observe's que "i1" ocorre dos segons després de cada ocurrència de l'esdeveniment "i2" i "i2" es dona tres segons després de cada ocurrència de "i1". Per tant, tots dos esdeveniments es donen cada cinc segons.
d	L'esdeveniment "i1" ocorre cada tres segons. Fals. Ocorre cada cinc segons.

13. Considerant el programa de la qüestió anterior, seleccione l'afirmació correcta:

a	No es mostrarà cap missatge durant la seua execució.
----------	--

A

	Fals. Cada vegada que es donen "i1" o "i2" es mostrarà un missatge indicant quantes vegades s'ha donat l'esdeveniment.
b	No es genera cap esdeveniment durant la seua execució, ja que les crides a emit() són incorrectes. Fals. Totes les crides a emit() s'han utilitzat correctament i els esdeveniments s'estan generant sense cap problema.
c	No es pot tenir més d'un listener per a un mateix esdeveniment. Per tant, el programa avorta immediatament, generant una excepció. Fals. Un esdeveniment pot tenir tants listeners com el programador decidisca. No es genera cap excepció a causa d'això.
d	El programa reporta correctament cada esdeveniment generat. Vertader. Això és conseqüència de tot el que s'ha justificat en cadascun dels apartats d'aquesta qüestió i l'anterior.

14. En ØMQ, en utilitzar un patró de comunicacions REQ-REP, se sol utilitzar bind() en el socket REP i connect() en el REQ, però altres variants viables són...

a	Es pot fer bind() sobre tots dos sockets (REQ i REP) utilitzant la mateixa adreça i port en ambdues crides. Fals. En aquest cas s'obtindria un error per haver utilitzat una adreça que ja estava en ús i la segona crida a bind() no funcionaria.
b	Es pot fer bind() en el socket REQ i connect() en el REP. Vertader. Aquesta és una alternativa que pot funcionar. Va arribar a ser utilitzada en algunes activitats del Seminari 3.
c	No és necessari cap bind(). N'hi ha prou amb realitzar un connect() en tots dos sockets sobre la mateixa adreça i port. Fals. Un dels sockets del canal ha d'utilitzar la crida bind(). Per a construir un canal, un dels sockets ha de realitzar bind(), o bindSync(), i l'altre (o uns altres) connect().
d	No s'admet cap variant. Sempre hem de començar amb un bind() per al socket REP i després fer un connect() en el socket REQ. Fals. Això contradiu el que s'ha dit en l'opció "b" i aquella és l'opció correcta.

15. Considerant aquests dos programes...

<pre>// server.js var net = require('net'); var server = net.createServer(function(c) { console.log('server connected'); c.on('end', function() { console.log('server disconnected'); }); c.on('data', function(data) { console.log('Request: ' + data); c.write(data + 'World!'); }); }); server.listen(9000);</pre>	<pre>// client.js var net = require('net'); var client = net.connect({port: 9000}, function() { client.write('Hello '); }); client.on('data', function(data) { console.log('Reply: ' + data); client.end(); }); client.on('end', function() { console.log('client ' + 'disconnected');</pre>
--	--

Selecione l'opció correcta:

a	El client acaba després d'enviar una petició i rebre la seua resposta. No emetrà més peticions. Vertader. Per a fer això, tanca la seua connexió amb el servidor utilitzant end()
----------	--

A

	després de rebre la resposta al seu únic missatge de petició.
b	El servidor acaba després d'enviar la seua primera resposta al primer client. Fals. La funció que ha utilitzat el programador com a <i>callback</i> per a <i>createServer()</i> pot manejar una connexió. Aquest <i>callback</i> s'utilitza tantes vegades com a connexions arriben a establir-se amb clients. A més, el servidor no acabarà. Romandrà en funcionament esperant noves connexions.
c	El servidor solament pot gestionar una connexió. Fals. La gestió de connexions ha sigut explicada en la justificació de l'apartat anterior.
d	Aquest client no pot connectar-se a aquest servidor. Fals. Pot connectar-se sense problemes si el servidor ha sigut iniciat correctament quan el client ho intenta.

16. Els algorismes d'elecció de líder (del Seminari 2)...

a	...no tenen condicions de seguretat. Fals. Tot algorisme distribuït ha de tenir almenys una condició de seguretat. De fet, es van presentar quatre condicions de seguretat per a aquests algorismes: (1) el líder ha de ser únic, (2) tots els processos han de triar el mateix líder, (3) una vegada triat, el líder no canvia mentre no hi haja fallades, (4) una vegada triat, els participants poden acabar l'execució de l'algorisme.
b	...solen utilitzar-se quan el coordinador actual d'un altre algorisme falla. Vertader. Aquest és l'esdeveniment que sol iniciar aquests algorismes.
c	...assumeixen, en els seus models de sistema, que no hi haurà fallades en el sistema. Fals. Si no hi haguera fallades no hi hauria cap necessitat de triar un nou líder.
d	...necessiten una imatge consistent de l'estat global actual del sistema (presa, per exemple, amb l'algorisme de Chandy i Lamport) per a poder iniciar-se. Fals. Cada procés participant gestiona les seues pròpies variables. No es necessita utilitzar cap estat global, acordat entre tots els participants, per a començar.

17. Suposem que es necessita implantar un servei de xat utilitzant node.js i ØMQ. El servidor difon els missatges dels usuaris i mai ha de suspendre's tractant d'enviar un missatge. 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 necessita un <i>socket</i> DEALER i un altre ROUTER per a equilibrar la càrrega entre els clients. Fals. Cap <i>socket</i> d'aquests tipus permet difondre missatges als clients. En lloc d'aquests, el servidor necessitarà un <i>socket</i> PUB per a difondre els missatges que ha de propagar i un <i>socket</i> PULL per a rebre els missatges abans de la seua propagació.
b	Cada client necessita un <i>socket</i> PULL per a interactuar amb el servidor. Fals. El <i>socket</i> a emprar per a rebre missatges ha de ser un <i>socket</i> SUB.
c	Cada client necessita un <i>socket</i> REP per a interactuar amb el servidor. Fals. El <i>socket</i> a emprar per a rebre missatges ha de ser un <i>socket</i> SUB.
d	Cada client necessita un <i>socket</i> SUB per a interactuar amb el servidor. Vertader. El <i>socket</i> a emprar per a rebre missatges ha de ser un <i>socket</i> SUB.

A

18. Considerant aquests programes...

<pre>//client.js var zmq=require('zmq'); var so=zmq.socket('dealer'); so.connect('tcp://127.0.0.1:8888'); so.send('request'); so.on('message',function(req,rep){ console.log("%s %s",req,rep); });</pre>	<pre>// server.js var zmq = require('zmq'); var rp = zmq.socket('rep'); rp.bindSync('tcp://127.0.0.1:8888'); rp.on('message', function(msg) { console.log('Request: ' + msg); rp.send([msg, 'answer']); });</pre>
--	---

Selecione l'opció correcta:

a	<p>El servidor no pot rebre i processar els missatges d'aquest client.</p> <p>Vertader. El servidor utilitza un socket REP. Els sockets REP assumeixen que els missatges entrants tindran un delimitador buit en algun dels seus segments inicials. El client utilitza un socket DEALER per a interactuar amb el servidor i els missatges que està enviant amb ell no tenen cap delimitador explícit. Com els DEALER no afegien cap delimitador (cosa que sí fan els REQ, de manera implícita), això implica que els missatges no podran ser rebuts per aquest servidor.</p>
b	<p>El servidor pot obtenir els missatges enviats per aquest client si el servidor s'ha iniciat abans de llançar el client.</p> <p>Fals. Amb els programes mostrats, no és possible cap comunicació.</p>
c	<p>El client mostra "request answer" en la pantalla després d'enviar el seu missatge 'request'.</p> <p>Fals. Amb els programes mostrats, no és possible cap comunicació.</p>
d	<p>El servidor no pot funcionar. En lloc de bindSync() hauria d'utilitzar-se bind() per a corregir aquest problema.</p> <p>Fals. Amb els programes mostrats, no és possible cap comunicació. Com s'ha justificat en l'explicació de l'apartat "a", aquesta situació no està relacionada amb les operacions bind() o bindSync().</p>

19. Considerant aquests programes...

<pre>//client.js var zmq=require('zmq'); var rq=zmq.socket('dealer'); rq.connect('tcp://127.0.0.1:8888'); for (var i=1; i<100; i++) { rq.send(''+i); console.log("Sending %d",i); } rq.on('message',function(req,rep){ console.log("%s %s",req,rep); });</pre>	<pre>// server.js var zmq = require('zmq'); var rp = zmq.socket('dealer'); rp.bindSync('tcp://127.0.0.1:8888'); rp.on('message', function(msg) { var j = parseInt(msg); rp.send([msg, (j*3).toString()]); });</pre>
---	---

Selecione l'opció correcta:

a	<p>Client i servidor intercanvien missatges de manera sincrònica en aquest exemple, ja que tots dos segueixen un patró petició/resposta.</p> <p>Fals. A pesar que s'envien múltiples peticions des del client i es retornen múltiples respostes des del servidor, la comunicació no és sincrònica posat que tots dos processos utilitzen sockets DEALER. Els sockets DEALER són asincrònics. Per això, el client podria haver enviat totes les seues peticions abans de rebre la primera resposta. Això succeiria, per exemple, en cas que el servidor ja estiguera saturat amb els missatges enviats per altres clients.</p>
----------	---

A

b	El servidor respon amb un missatge de dos segments al client. El segon segment conté un valor que és tres vegades major que el del primer segment. Vertader. Aquest és el comportament implantat en el <i>listener</i> per a l'esdeveniment "message" del programa servidor.
c	El client envia 100 peticions al servidor. Fals. Solament envia 99 peticions.
d	El client mostra en pantalla aquesta seqüència: "Sending 1", "1 3", "Sending 2", "2 6", "Sending 3", "3 9"... Fals. Com ja s'ha explicat en la justificació de l'apartat "a" res garanteix que cada resposta s'emplace entre dos missatges de petició consecutius. Per tant, els missatges "Sending..." i les respostes mostrades no arriben a seqüenciar-se tal com suggereix l'enunciat d'aquest apartat.

20. Considere quina de les següents variacions generarà nous programes amb el mateix comportament que els de la pregunta anterior:

a	El socket 'rq' és de tipus 'REQ' i el 'rp' de tipus 'REP'. Fals. En aquest cas s'obtindria un comportament sincrònic, en lloc de l'asincrònic mostrat en la pregunta 19.
b	El socket 'rq' és de tipus 'PUSH' i el 'rp' de tipus 'PULL'. Fals. En aquest cas només seria possible una comunicació unidireccional (del client al servidor), sense admetre cap resposta del servidor.
c	Tots dos processos necessiten dos sockets, definint un canal PUSH->PULL entre client i servidor (peticions) i un altre PUSH->PULL de servidor a client (respostes). Vertader. Així s'implantarien dos canals unidireccionals i asincrònics, emulant de manera apropiada el comportament d'un canal DEALER-DEALER.
d	El socket 'rq' és de tipus 'PUB' i el 'rp' de tipus 'ROUTER'. Fals. Els sockets PUB són unidireccionals. No admeten la recepció de cap missatge. Per tant, els clients no podrien obtenir cap resposta.