A

Este examen contiene 20 cuestiones de opción múltiple. En cada pregunta solo una de sus respuestas es correcta. Las contestaciones deben presentarse en una hoja entregada aparte. Las respuestas correctas aportan 0.5 puntos mientras que las incorrectas restan 0.167 puntos.

TEORÍA

1. El modelo de servicio PaaS para la computación en la nube...

a	…tiene como principal objetivo la virtualización del *hardware*. Falso. La virtualización del *hardware* es un aspecto que debe ser gestionado en el nivel de infraestructura. El modelo de "plataforma como servicio" (PaaS) se sitúa sobre el nivel de gestión de infraestructura (que sería el modelo IaaS en un entorno de computación en la nube).
b	…proporciona un servicio de aplicaciones distribuidas a sus usuarios directos. Falso. El modelo de "*software* como servicio" (SaaS) es el responsable de esa clase de servicios. El modelo PaaS se ubica por debajo del SaaS en una arquitectura de servicios estructurada en niveles.
c	…gestiona la infraestructura subyacente y automatiza el despliegue y la administración de aplicaciones para los proveedores de éstas. Verdadero. Esas son las tareas a considerar en el modelo PaaS.
d	…evita en lo posible la concurrencia. Falso. Los modelos de servicio para la computación en la nube no deben evitar la concurrencia. Todos los sistemas distribuidos son ejemplos de sistemas concurrentes. La concurrencia forma parte de cualquier servicio distribuido. Por tanto, los modelos de servicio deben ser concurrentes.

2. La Wikipedia…

a	…es un ejemplo de aplicación distribuida que utiliza la interacción "peer-to-peer". Por tanto, es un servicio fácilmente escalable. Falso. La Wikipedia utiliza interacciones cliente/servidor.
b	…utiliza los navegadores web como un tipo específico de agente servidor. Falso. Los navegadores web son ejemplos de agentes clientes.
c	…almacena su información persistente en los nodos clientes. Falso. Su información persistente se mantiene en bases de datos en el dominio servidor, no en los nodos clientes.
d	…utiliza *proxies* inversos para mejorar su rendimiento. Verdadero. Esa es una de las técnicas explicadas en el Tema 1 para mejorar el rendimiento y la escalabilidad. Los *proxies* inversos han sido utilizados en la Wikipedia desde las primeras ediciones de ese sistema.

3. LAMP son las siglas de:

a	*Light and Available Multi-Processing* (es decir, multiprocesamiento ligero y disponible). Falso.

b	Linux, Apache, MySQL y PHP. Verdadero. Los sistemas LAMP se explicaron en el Tema 1 (en la sección dedicada a la Wikipedia) para describir cómo son las arquitecturas habituales de los servidores web.
c	Linux, Acrobat, MongoDB y Python. Falso.
d	Linux, Apache, Memcache y PostgreSQL. Falso.

4. Un modelo de sistema...

a	...establece la arquitectura del sistema distribuido. Falso. La arquitectura de un sistema o servicio distribuido no se llega a definir en su modelo de sistema. Los modelos de sistema no consideran esos aspectos.
b	...describe cuidadosamente los equipos y protocolos de comunicación que deben usarse en un sistema distribuido real. Falso. Los equipos de comunicación no se describen en un modelo de sistema.
c	...facilita una imagen de alto nivel de un sistema, especificando las propiedades que deben considerarse al escribir algoritmos en ese sistema. Verdadero. Este sería un resumen válido de lo que debe considerarse en un modelo de sistema para un sistema distribuido, según lo que se explicó en el Tema 2.
d	...especifica todos los detalles y algoritmos que están siendo utilizados en un *middleware.* Falso. Un modelo de sistema no proporciona ningún detalle sobre partes del *software*. Solo las propiedades principales llegan a considerarse y eso se hace a un alto nivel de abstracción. Esas propiedades deben ser relevantes a la hora de escribir los algoritmos que proporcionen alguna solución a los problemas existentes en ese entorno.

5. Sobre el tiempo y la sincronización en sistemas distribuidos:

a	Un modelo de sistema conviene que sea sincrónico ya que la sincronía evita las condiciones de carrera. Falso. No todos los tipos de sincronía guardan relación con las condiciones de carrera. Además, cuanta más sincronización se asuma en un modelo de sistema, mayor dificultad habrá para facilitar ese comportamiento asumido en el modelo de sistema.
b	La sincronización debe evitarse tanto como sea posible pues compromete el rendimiento y la escalabilidad. Verdadero. Los mecanismos de sincronización bloquearán la actividad de los agentes en algunos casos. Por ello, no son convenientes para alcanzar niveles altos de rendimiento y escalabilidad.
c	La comunicación sincrónica nunca bloquea a los procesos. Falso. La comunicación sincrónica bloquea a los procesos emisores mientras no reciban algún tipo de respuesta por parte de los receptores.
d	Los relojes locales de cada nodo pueden sincronizarse sin intercambiar mensajes. Para ello se utiliza el algoritmo de Cristian. Falso. El algoritmo de Cristian está basado en el intercambio de mensajes. Todos los algoritmos de sincronización de relojes necesitan que cada proceso interactúe con algún agente externo.

A

6. En el paradigma de programación asincrónica...

a	Hay múltiples hilos en cada proceso. Falso. No es necesario tener múltiples hilos en cada proceso en ese modelo de programación. Node.js es un ejemplo donde no hay varios hilos por proceso.
b	Todos los eventos ocurren simultáneamente. Falso. Nada obliga a que todos los eventos se den a la vez. Cada evento puede ocurrir en un instante distinto.
c	Si múltiples eventos ocurren a la vez, entonces sus acciones estarán habilitadas pero se ejecutarán secuencialmente y con garantías de atomicidad. Verdadero. Todas las acciones asociadas con eventos son atómicas (esto es, se ejecutan en un solo turno y no pueden ser interrumpidas) y, debido a ello, serán ejecutadas una tras otra (es decir, secuencialmente).
d	Los programas no pueden estar basados en *callbacks*. Falso. Pueden estar basados en *callbacks* y, de hecho, en esta asignatura hemos utilizado un lenguaje de programación (JavaScript) que utiliza esa aproximación.

7. Todos los sistemas (*middleware*) de mensajería...

a	...son ejemplos de servicio de autenticación (es decir, un servicio relacionado con la seguridad). Falso. Los servicios de autenticación no pueden considerarse, en el caso general, un ejemplo de sistema de mensajería.
b	...no están obligados a facilitar transparencia de ubicación. Verdadero. La comunicación orientada a mensajes tiene como principal objetivo un buen rendimiento. La transparencia de ubicación no es un objetivo en estos sistemas. Hemos trabajado con un ejemplo en TSR: ØMQ. En ese *middleware* las operaciones bind() y connect() no proporcionan transparencia de ubicación al programador ya que este debe especificar la dirección IP y el puerto en cada una de esas operaciones. Para proporcionar comunicación con transparencia de ubicación se deberían utilizar nombres (en lugar de direcciones) para referirse a otros agentes.
c	...utilizan *proxies* y esqueletos para mejorar su rendimiento. Falso. Los *proxies* y esqueletos se utilizan en los mecanismos de invocación remota de métodos, facilitando las mismas interfaces que los agentes u objetos remotos con los que deba interactuarse. Los sistemas de mensajería no utilizan ese tipo de interfaz. Normalmente, usan operaciones send() proporcionadas por *sockets*.
d	...utilizan algoritmos centralizados y dependen de un gestor de comunicaciones (*broker*) central. Falso. Los sistemas de mensajería pueden tener un gestor de comunicaciones o no tenerlo. Por tanto, no dependen en todos los casos de ese tipo de gestor. Incluso en el caso de que se utilizara un gestor, este no necesitará muchos pasos (es decir, un algoritmo que no sea trivial) para gestionar la comunicación entre diferentes agentes. Además, los sistemas de mensajería más eficientes no suelen tener gestor, como hemos visto en el caso de ØMQ.

8. La comunicación no persistente basada en mensajes…

a	…bloquea al servidor mientras el receptor no haya notificado que ha podido entregar el mensaje. Falso. Esa no es la definición de comunicación no persistente sino la de la comunicación sincrónica.
b	…facilita transparencia de ubicación. Falso. El grado de persistencia en un sistema de comunicaciones no está relacionado con la transparencia de ubicación. Para conseguir transparencia de ubicación necesitamos *stubs* en ambos agentes (emisor y receptor) proporcionando interfaces locales idénticas a las del agente remoto.
c	…garantiza transparencia de fallos. Falso. El grado de persistencia en un sistema de comunicaciones no está relacionado con la transparencia de fallos. Para conseguir transparencia de fallos deberíamos replicar los agentes emisores y receptores (para ocultar los fallos en los procesos) y reintentar las acciones de envío (para soportar la pérdida de mensajes).
d	…no permite que la comunicación se inicie hasta que emisor y receptor puedan gestionar la transmisión del mensaje. Verdadero. La comunicación no persistente utiliza canales que no pueden guardar los mensajes que están siendo propagados. Por ello, ambos agentes de comunicación deben estar preparados antes de iniciar la transmisión de cualquier mensaje.

SEMINARIOS

9. Si consideramos este programa...

```
var fs=require('fs');
if (process.argv.length<5) {
      console.error('Se necesitan más nombres de ficheros!!');
      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('Fichero '+name+': '+data.length+' bytes.');
      })
})
console.log('Se han procesado '+files.length+' ficheros.');
```

Seleccione la opción correcta, asumiendo que no habrá errores en su ejecución:

a	El programa necesita recibir al menos 5 nombres de fichero como argumentos desde la línea de órdenes. Falso. Necesita al menos 5 argumentos pero el primero de ellos es siempre “node” (el nombre del intérprete) y el segundo el nombre del programa a ejecutar. Por tanto, solo se está solicitando que haya 3 nombres de fichero como mínimo (en lugar de 5, que es lo que se dice en el enunciado).

b	El programa muestra el nombre y el tamaño de cada uno de los ficheros recibidos como argumentos. Verdadero. Esta es la funcionalidad facilitada por este programa.
c	La última línea que imprime es "**Se han procesado N ficheros.**", siendo N el número de ficheros recibidos como argumentos. Falso. La sentencia que muestra ese mensaje está al final del programa pero JavaScript en un lenguaje de programación asincrónico. Por ello, esa sentencia llega a ejecutarse en el primer turno mientras que los demás console.log() (ubicados en el *callback* de la función readFile()) se ejecutarán en turnos posteriores. Esto implica que el mensaje mencionado en este enunciado se imprimirá al empezar la ejecución del programa (es decir, será la primera cosa que se muestre en pantalla, no la última).
d	El programa puede ser ejecutado en un solo turno. Falso. Necesitará múltiples turnos, como se ha descrito en el apartado "c".

10. Considerando el programa visto en la pregunta anterior, seleccione la opción correcta:

a	No podrá escribir el nombre de fichero apropiado en cada iteración. En su lugar, siempre imprime el nombre del último fichero. Falso. El nombre de fichero se ha gestionado correctamente en el *callback* de la función readFile() ya que esa función está, a su vez, en el *callback* del bucle forEach() y el nombre de fichero se está recibiendo como argumento en esa función más externa. Así, el esquema resultante es similar a una clausura y garantiza que cada invocación del *callback* de readFile se haga en un ámbito que gestiona el valor apropiado para la variable "name".
b	Genera una excepción y aborta si ocurre algún error cuando trata de leer algún fichero. Falso. Si se diera algún error en ese punto, se comunicaría mediante el parámetro "error" del *callback* de readFile(). En ese caso se mostrará un mensaje de error y no se generará ninguna excepción.
c	En caso de error, muestra un mensaje en la pantalla y continúa. Verdadero. Ya se ha explicado en la justificación del apartado anterior.
d	Muestra el mismo tamaño de fichero en todas las iteraciones. Necesitaría una clausura para corregir ese defecto. Falso. Como ya se ha explicado en el apartado "a", ya se está utilizando una clausura en esta solución.

11. Algunas condiciones de seguridad para todos los algoritmos de exclusión mutua vistos en el Seminario 2 son...

a	Un proceso accede a la sección crítica tras completar su protocolo de salida. Falso. Un proceso completa su protocolo de salida cuando ha abandonado su sección crítica. Por tanto, cuando eso suceda estará en su sección restante (es decir, fuera de la sección crítica).

b	Los procesos deben utilizar canales de comunicación asincrónicos. Falso. Ninguno de los algoritmos presentados en el seminario necesitaban canales de comunicación asincrónicos.
C	No puede haber más de un proceso ejecutando la sección crítica. Verdadero. Esta es la condición de corrección aplicable a todas las soluciones para este problema. Las condiciones de seguridad deben establecer las restricciones que deben ser respetadas por las soluciones correctas. Esto es, las condiciones de seguridad garantizan que nada incorrecto o inválido ocurra mientras se estén ejecutando las tareas descritas en un algoritmo.
d	No inanición: debe limitarse el número de veces que un proceso en su protocolo de entrada ceda el paso a otros procesos. Falso. Esta es una condición de vivacidad. No es una condición de seguridad.

12. Si consideramos este programa...

```
var ev = require('events');
var emitter = new ev.EventEmitter;
var num1 = 0;
var num2 = 0;
function emit_e1() { emitter.emit("e1") }
function emit_e2() { emitter.emit("e2") }
emitter.on("e1", function() {
   console.log( "El evento e1 ha ocurrido " + ++num1 + " veces.")});
emitter.on("e2", function() {
   console.log( "El evento e2 ha ocurrido " + ++num2 + " veces.")});
emitter.on("e1", function() {
   setTimeout( emit_e2, 3000 )});
emitter.on("e2", function() {
   setTimeout( emit_e1, 2000 )});
setTimeout( emit_e1, 2000 );
```

Seleccione la afirmación correcta:

a	El evento "e1" ocurre solo una vez, dos segundos después de que el programa sea iniciado. Falso. El evento "e1" ocurre por primera vez dos segundos después de haberse iniciado el programa. Posteriormente, ocurre cada cinco segundos. Por tanto, no ocurre solo una vez.
b	El evento "e2" nunca ocurre. Falso. El evento "e2" ocurre tres segundo después de cada ocurrencia del evento "e1". Por tanto, ocurrirá (aproximadamente) tantas veces como ocurra "e1".
C	El evento "e2" ocurre cada cinco segundos. Verdadero. Obsérvese que "e1" ocurre dos segundos después de cada ocurrencia del evento "e2" y "e2" se da tres segundos después de cada ocurrencia de "e1". Por tanto, ambos eventos se dan cada cinco segundos.
d	El evento "e1" ocurre cada tres segundos. Falso. Ocurre cada cinco segundos.

13. Considerando el programa de la cuestión anterior, seleccione la afirmación correcta:

a	No se mostrará ningún mensaje durante su ejecución. Falso. Cada vez que se den "e1" o "e2" se mostrará un mensaje indicando cuántas veces se ha dado ese evento.

b	No se genera ningún evento durante su ejecución, pues las llamadas a emit() son incorrectas. Falso. Todas las llamadas a emit() se han utilizado correctamente y los eventos se están generando sin ningún problema.
c	No se puede tener más de un *listener* para un mismo evento. Por tanto, el programa aborta inmediatamente, generando una excepción. Falso. Un evento puede tener tantos *listeners* como el programador prefiera. No se genera ninguna excepción debido a ello.
d	El programa reporta correctamente cada evento generado. Verdadero. Esto es consecuencia de todo lo que se ha justificado en cada uno de los apartados de esta cuestión y la anterior.

14. En ØMQ, al utilizar un patrón de comunicaciones REQ-REP, se suele utilizar bind() en el *socket* REP y connect() en el REQ, pero otras variantes viables son...

a	Se puede hacer bind() sobre ambos *sockets* (REQ y REP) utilizando la misma dirección y puerto en ambas llamadas. Falso. En ese caso se obtendría un error por haber utilizado una dirección que ya estaba en uso y la segunda llamada a bind() no funcionaría.
b	Se puede hacer bind() en el *socket* REQ y connect() en el REP. Verdadero. Esta es una alternativa que puede funcionar. Llegó a ser utilizada en algunas actividades del Seminario 3.
c	No es necesario ningún bind(). Basta con realizar un connect() en ambos *sockets* sobre la misma dirección y puerto. Falso. Uno de los sockets del canal debe utilizar la llamada bind(). Para construir un canal, uno de los sockets debe realizar bind(), o bindSync(), y el otro (u otros) connect().
d	No se admite ninguna variante. Siempre tenemos que empezar con un bind() para el *socket* REP y después hacer un connect() en el *socket* REQ. Falso. Esto contradice lo dicho en la opción "b" y aquella es la opción correcta.

15. Considerando estos dos programas...

```
// server.js
var net = require('net');
var server = net.createServer(
 function(c) {//'connection' listener
  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);
```

```
// 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');
});
```

Seleccione la opción correcta:

a	El cliente termina tras enviar una petición y recibir su respuesta. No emitirá más peticiones. Verdadero. Para ello, cierra su conexión con el servidor utilizando end() tras recibir la respuesta a su único mensaje de petición.

b	El servidor termina tras enviar su primera respuesta al primer cliente. Falso. La función que ha utilizado el programador como *callback* para createServer() puede manejar una conexión. Ese *callback* se utiliza tantas veces como conexiones lleguen a establecerse con clientes. Además, el servidor no terminará. Permanecerá en funcionamiento esperando nuevas conexiones.
c	El servidor solo puede manejar una conexión. Falso. La gestión de conexiones ha sido explicada en la justificación del apartado anterior.
d	Este cliente no puede conectarse a este servidor. Falso. Puede conectarse sin problemas si el servidor ha sido iniciado correctamente cuando el cliente lo intenta.

16. Los algoritmos de elección de líder (del Seminario 2)...

a	...no tienen condiciones de seguridad. Falso. Todo algoritmo distribuido debe tener al menos una condición de seguridad. De hecho, se presentaron cuatro condiciones de seguridad para estos algoritmos: (1) el líder debe ser único, (2) todos los procesos deben elegir al mismo líder, (3) una vez elegido, el líder no cambia mientras no haya fallos, (4) una vez elegido, los participantes pueden finalizar la ejecución del algoritmo.
b	...suelen utilizarse cuando el coordinador actual de otro algoritmo falla. Verdadero. Ese es el evento que suele iniciar estos algoritmos.
c	...asumen, en sus modelos de sistema, que no habrá fallos en el sistema. Falso. Si no hubiera fallos no habría ninguna necesidad de elegir un nuevo líder.
d	...necesitan una imagen consistente del estado global actual del sistema (tomada, por ejemplo, con el algoritmo de Chandy y Lamport) para poder iniciarse. Falso. Cada proceso participante gestiona sus propias variables. No se necesita utilizar ningún estado global, acordado entre todos los participantes, para empezar.

17. Supongamos que se necesita implantar un servicio de chat utilizando node.js y ØMQ. El servidor difunde los mensajes de los usuarios y nunca debe suspenderse tratando de enviar un mensaje. Los programas clientes envían los mensajes de los usuarios al servidor, esperan los mensajes reenviados por el servidor e informan al servidor cuando un usuario se incorpora o abandona el sistema. Para implantar este servicio de chat...

a	El servidor necesita un *socket* DEALER y otro ROUTER para equilibrar la carga entre los clientes. Falso. Ningún socket de estos tipos permite difundir mensajes a los clientes. En lugar de estos, el servidor necesitará un socket PUB para difundir los mensajes que debe propagar y un socket PULL para recibir los mensajes antes de su propagación.
b	Cada cliente necesita un *socket* PULL para interactuar con el servidor. Falso. El socket a emplear para recibir mensajes debe ser un socket SUB.
c	Cada cliente necesita un *socket* REP para interactuar con el servidor. Falso. El socket a emplear para recibir mensajes debe ser un socket SUB.
d	Cada cliente necesita un *socket* SUB para interactuar con el servidor. Verdadero. El socket a emplear para recibir mensajes debe ser un socket SUB.

18. Considerando estos programas...

```
//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);
});
```

```
// 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']);
});
```

Seleccione la opción correcta:

a	El servidor no puede recibir y procesar los mensajes de este cliente. Verdadero. El servidor utiliza un socket REP. Los sockets REP asumen que los mensajes entrantes tendrán un delimitador vacío en alguno de sus segmentos iniciales. El cliente utiliza un socket DEALER para interactuar con el servidor y los mensajes que está enviando con él no tienen ningún delimitador explícito. Como los DEALER no añaden ningún delimitador (cosa que sí hacen los REQ, de manera implícita), esto implica que los mensajes no podrán ser recibidos por este servidor.
b	El servidor puede obtener los mensajes enviados por este cliente si el servidor se ha iniciado antes de lanzar el cliente. Falso. Con los programas mostrados, no es posible ninguna comunicación.
c	El cliente muestra “request answer” en la pantalla tras enviar su mensaje ‘request’. Falso. Con los programas mostrados, no es posible ninguna comunicación.
d	El servidor no puede funcionar. En lugar de bindSync() debería utilizarse bind() para corregir este problema. Falso. Con los programas mostrados, no es posible ninguna comunicación. Como se ha justificado en la explicación del apartado “a”, esa situación no está relacionada con las operaciones bind() o bindSync().

19. Considerando estos programas...

```
//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);
});
```

```
// 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()]);
});
```

Seleccione la opción correcta:

a	Cliente y servidor intercambian mensajes de manera sincrónica en este ejemplo, pues ambos siguen un patrón petición/respuesta. Falso. A pesar de que se envíen múltiples peticiones desde el cliente y se retornen múltiples respuestas desde el servidor, la comunicación no es sincrónica puesto que ambos procesos utilizan sockets DEALER. Los sockets DEALER son asincrónicos. Por ello, el cliente podría haber enviado todas sus peticiones antes de recibir la primera respuesta. Esto sucedería, por ejemplo, en caso de que el servidor ya estuviera saturado con los mensajes enviados por otros clientes.

b	El servidor responde con un mensaje de dos segmentos al cliente. El segundo segmento contiene un valor que es tres veces mayor que el del primer segmento. Verdadero. Este es el comportamiento implantado en el *listener* para el evento "message" del programa servidor.
c	El cliente envía 100 peticiones al servidor. Falso. Solo envía 99 peticiones.
d	El cliente muestra en pantalla esta secuencia: "Sending 1", "1 3", "Sending 2", "2 6", "Sending 3", "3 9"... Falso. Como ya se ha explicado en la justificación del apartado "a" nada garantiza que cada respuesta se emplace entre dos mensajes de petición consecutivos. Por tanto, los mensajes "Sending..." y las respuestas mostradas no tienen por qué secuenciarse tal como sugiere el enunciado de este apartado.

20. Considere cuál de las siguientes variaciones generará nuevos programas con el mismo comportamiento que los de la pregunta anterior:

a	El *socket* 'rq' es de tipo 'REQ' y el 'rp' de tipo 'REP'. Falso. En ese caso se obtendría un comportamiento sincrónico, en lugar del asincrónico mostrado en la pregunta 19.
b	El *socket* 'rq' es de tipo 'PUSH' y el 'rp' de tipo 'PULL'. Falso. En ese caso solo sería posible una comunicación unidireccional (del cliente al servidor), sin admitir ninguna respuesta del servidor.
c	Ambos procesos necesitan dos *sockets*, definiendo un canal PUSH->PULL entre cliente y servidor (peticiones) y otro PUSH->PULL de servidor a cliente (respuestas). Verdadero. Así se implantarían dos canales unidireccionales y asincrónicos, emulando de manera apropiada el comportamiento de un canal DEALER-DEALER.
d	El *socket* 'rq' es de tipo 'PUB' y el 'rp' de tipo 'ROUTER'. Falso. Los sockets PUB son unidireccionales. No admiten la recepción de ningún mensaje. Por tanto, los clientes no podrían obtener ninguna respuesta.