TSR: Primer Parcial

El examen consta de 20 preguntas de opción múltiple. En cada caso solo una de las respuestas es correcta. Se debe responder en una hoja aparte. En caso de respuesta correcta, ésta aportará 0.5 puntos. En caso de error, la contribución es negativa: -0.167.

TEORÍA

1. Considerando el Tema 1, estas afirmaciones describen correctamente algunos aspectos de los sistemas distribuidos:

a	Todo sistema concurrente es un sistema distribuido.
b	El servicio de correo electrónico es un ejemplo de sistema distribuido.
c	Los agentes en un sistema distribuido no pueden compartir ningún recurso pues están ubicados en ordenadores diferentes.
d	El programador de una aplicación distribuida no necesita preocuparse sobre la tolerancia a fallos, pues ya está garantizada implícitamente por el sistema distribuido.

2. Una de las razones para decir que la Wikipedia es una aplicación distribuida escalable es..

a	Desde su primera edición, se ha implantado en la nube siguiendo el modelo de servicio SaaS.
b	Es un sistema LAMP, y todos los sistemas de este tipo son escalables.
c	Utiliza interacción P2P y esto mejora su escalabilidad.
d	Utiliza *proxies* inversos (como cachés) y replicación de componentes.

3. El objetivo principal del modelo de servicio PaaS es…

a	Automatizar la configuración, despliegue y actualización de servicios distribuidos y su adaptación a cargas variables.
b	Automatizar la provisión de infraestructura.
c	Proporcionar servicios distribuidos bajo un modelo de pago a medida (“pay-as-you-go”).
d	Gestionar datos persistentes bajo un modelo de pago a medida (“pay-as-you-go”).

4. El tema 2 propone un modelo de sistema distribuido sencillo porque ese modelo…

a	…garantiza persistencia de datos.
b	…es necesario para comparar dos tipos de programación: asíncrona y multi-hilo.
c	…proporciona una buena base para diseñar algoritmos distribuidos y para razonar acerca de su corrección antes de iniciar su desarrollo.
d	…demuestra que las situaciones de bloqueo de actividades impiden que los servicios escalen.

5. Esta es la mejor solución para proporcionar persistencia de datos:

a	Usar servidores *stateful* (es decir, servidores con gestión de estado en sus interacciones).
b	Replicar los datos.
c	Usar los discos duros más fiables.
d	Evitar los accesos concurrentes a los datos.

6. El modelo de sistema sencillo descrito en el Tema 2 está soportado directamente por la programación asíncrona porque…

a	…la programación asíncrona está basada en comunicación causal.
b	…los procesos en el modelo de sistema sencillo son multi-hilo.
c	…hay una traducción directa entre guardas + acciones en el modelo y eventos + *callbacks* en la programación asíncrona.
d	…los procesos siguen implícitamente el modelo de fallos de parada en la programación asíncrona.

7. Para desarrollar aplicaciones escalables resulta más recomendable un sistema (middleware) de mensajería (MoM, por sus siglas en inglés) que la invocación de objetos remotos (RMI, en inglés) porque…

a	MoM proporciona transparencia de ubicación, pero RMI no puede proporcionarla.
b	MoM es inherentemente asincrónico, mientras RMI es sincrónico.
c	Los procesos que utilizan MoM asumen un espacio compartido de recursos. En RMI los procesos no pueden compartir recursos.
d	Los procesos que utilizan MoM están replicados automáticamente. En RMI, la replicación no está permitida.

8. Los sistemas (middleware) de mensajería persistente…

a	…proporcionan transparencia de ubicación.
b	…implantan la persistencia automáticamente al utilizar un servicio de nombres.
c	…pueden ser desarrollados de manera sencilla utilizando una implantación basada en gestores (“broker-based”).
d	…no pueden ser utilizados en comunicaciones asincrónicas.

TSR

SEMINARIOS

9. Considerando este programa:

```
var fs=require('fs');
if (process.argv.length<5) {
  console.error('More file names are needed!!');
  process.exit();
}
var files = process.argv.slice(2);
var i=-1;
do {
  i++;
  fs.readFile(files[i], 'utf-8', function(err,data) {
      if (err) console.log(err);
      else console.log('File '+files[i]+': '+data.length+' bytes.');
  })
} while (i<files.length-1);
console.log('We have processed '+files.length+' files.');
```

Estas afirmaciones son ciertas si se asume que ningún error aborta su ejecución:

a	Este programa muestra en todas las iteraciones, entre otra información, el nombre del último fichero facilitado en la línea de órdenes.
b	Muestra el nombre y tamaño de cada uno de los ficheros recibidos en la línea de órdenes.
c	Muestra "We have processed N files" al final de su ejecución, siendo N el número de nombres recibidos como argumentos.
d	Descarta los dos primeros nombres de fichero pasados como argumentos en la línea de órdenes.

10. Considerando el programa de la pregunta anterior…

a	Necesita múltiples turnos (de la cola de planificación del intérprete) para completar su ejecución, pues cada fichero utiliza su propio turno.
b	Genera una excepción y aborta si algún error ocurre cuando intenta leer un fichero.
c	El programa es incorrecto. Debería utilizar "var i=0" al inicializar la variable "i" para ser correcto.
d	Siempre muestra el mismo tamaño en todas las iteraciones. Necesitaríamos una clausura para evitar este comportamiento defectuoso.

11. En los algoritmos de exclusión mutua vistos en el Seminario 2 se puede afirmar que…

a	El algoritmo con servidor central minimiza el número de mensajes necesarios.
b	El algoritmo de anillo unidireccional tiene un retraso de sincronización de 1 mensaje.
c	El retraso de sincronización del algoritmo con multidifusión y relojes lógicos es de 2N-2 mensajes.
d	La versión del algoritmo de multidifusión y *quórums* descrita en la presentación satisface las 3 condiciones de corrección para estos algoritmos.

12. Considerando 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( "Event e1 has happened " + ++num1 + " times.")});
emitter.on("e2", function() {
   console.log( "Event e2 has happened " + ++num2 + " times.")});
emitter.on("e1", function() {
   setTimeout( emit_e2, 3000 )});
emitter.on("e2", function() {
   setTimeout( emit_e2, 2000 )});
setTimeout( emit_e1, 2000 );
```

Estas afirmaciones son ciertas:

a	El evento “e1” ocurre solo una vez, 2 segundos después del inicio del proceso.
b	El evento “e2” nunca ocurre.
c	El periodo de “e2” es cinco segundos.
d	El periodo de “e1” es tres segundos.

13. En el programa de la pregunta anterior…

a	El primer evento “e2” ocurre cinco segundos después de haber iniciado el proceso.
b	No se genera ningún evento en su ejecución, pues las llamadas a emit() son incorrectas.
c	No se puede tener más de un *listener* para cada evento. Por tanto, el proceso aborta generando una excepción en la tercera llamada a emitter.on().
d	Ninguno de sus eventos ocurre periódicamente.

14. El patrón de comunicación REQ-REP de ØMQ se considera sincrónico porque…

a	Sigue el patrón de interacción cliente/servidor y en ese patrón el cliente permanece bloqueado hasta que se reciba una respuesta.
b	Tanto REQ como REP son *sockets* bidireccionales, es decir, ambos pueden enviar y recibir mensajes.
c	La cola de envío del *socket* REQ tiene capacidad limitada. Solo puede mantener un mensaje.
d	Los *sockets* REQ no pueden transmitir una solicitud mientras la respuesta anterior no se haya recibido. Los *sockets* REP no pueden enviar una respuesta antes de la solicitud.

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 i=0;
var client = net.connect({port:
  9000}, function() {
    client.write('Hello ');
  });
client.on('data', function(data) {
  console.log('Reply: '+data);
  i++; if (i==2) client.end();
});
client.on('end', function() {
  console.log('client ' +
    'disconnected');
});
```

Las siguientes afirmaciones son ciertas:

a	El servidor termina tras enviar su primera respuesta al primer cliente.
b	El cliente nunca termina.
c	El servidor sólo puede manejar una conexión.
d	Este cliente no puede conectarse al servidor.

16. Los algoritmos de elección de líder (vistos en el Seminario 2)…

a	...no tienen condiciones de seguridad.
b	...pueden estar indefinidamente seleccionando un proceso líder.
c	...deben asegurar que un solo líder ha sido elegido.
d	...deben respetar el orden causal.

17. Asuma que se va a desarrollar un servicio de exclusión mutua utilizando NodeJS y ØMQ, empleando el primer algoritmo explicado en el Seminario 2: el basado en un servidor central. La mejor opción (de entre las mostradas) para ello sería…

a	El servidor utilizará un *socket* DEALER y un *socket* ROUTER para equilibrar la carga entre sus clientes.
b	Cada cliente utilizará un *socket* DEALER para interactuar con el servidor.
c	Cada cliente utilizará un *socket* REP para interactuar con el servidor.
d	Cada cliente utilizará un *socket* SUB para interactuar con el servidor.

18. Asuma que se va a desarrollar un servicio de exclusión mutua utilizando NodeJS y ØMQ, empleando el 2º algoritmo explicado en el Seminario 2: el de anillo (virtual) unidireccional. La mejor opción (de entre las mostradas) para ello sería…:

a	Utilizar algún algoritmo de elección de líder para seleccionar un proceso coordinador.
b	Todos los procesos tienen el mismo rol y necesitan un *socket* REP para enviar mensajes y un *socket* REQ para recibirlos.
c	Todos los procesos tienen el mismo rol y necesitan un *socket* PUSH para enviar el *token* y un *socket* PULL para recibirlo.
d	Todos los procesos tienen el mismo rol y necesitan un *socket* PUB para enviar el *token* y un *socket* DEALER para recibirlo.

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()]);
});
```

Las siguientes afirmaciones son ciertas:

a	El cliente y el servidor intercambian mensajes sincrónicamente, pues ambos siguen un patrón petición/respuesta.
b	El servidor retorna un mensaje con 2 segmentos al cliente. El segundo segmento contiene un valor 3 veces superior al del primer segmento.
c	El cliente y el servidor pueden ejecutarse en diferentes ordenadores. Interactúan sin problemas en ese caso.
d	El cliente no envía ningún mensaje pues la sentencia `''+i` genera una excepción y el proceso aborta en ese punto.

20. Considere cuál de las siguientes variaciones generaría nuevos programas con el mismo comportamiento observado en la pregunta 19 (A--> B significa que la sentencia A es reemplazada por la sentencia B).

a	El *socket* 'rq' será de tipo PULL y el *socket* 'rp' de tipo PUSH.
b	El *socket* 'rq' será de tipo PUSH y el *socket* 'rp' de tipo PULL.
c	Cliente: rq.connect('tcp://127.0.0.1:8888'); --> rq.bindSync('tcp://*:8888'); Servidor: rp.bindSync('tcp://127.0.0.1:8888'); --> rp.connect('tcp://127.0.0.1:8888');
d	El *socket* 'rq' será de tipo REP y el *socket* 'rp' de tipo REQ.