TSR - Primer Parcial. 2024-11-04

Este examen consta de 22 cuestiones, con una puntuación total de 10 puntos. Cada cuestión tiene 4 alternativas, de las cuales debe elegirse una sola opción. La nota se calcula de la siguiente forma: tras descartar las dos peores cuestiones, cada acierto suma 0.5 puntos y cada error descuenta 1/6 puntos. Debes contestar en la hoja de respuestas.

A

ALUMN@

1. ***En relación con las ventajas que aporta un sistema distribuido, selecciona la afirmación correcta***
   - **A.** Mejora el rendimiento
   - **B.** Permite mejorar la disponibilidad
   - **C.** Facilita la compartición de recursos
   - **D.** Todas las demás afirmaciones son ciertas

2. ***En relación con el concepto de cluster altamente disponible, dichos clusters:***
   - **A.** Sacrifican la disponibilidad de los servidores para mantener la integridad de la información gestionada
   - **B.** Requieren programas, equipos, y personal especializado dedicados, y por lo tanto resulta caro para las empresas
   - **C.** Ninguna de las demás afirmaciones es cierta
   - **D.** Sacrifican la integridad de la información gestionada para mantener la disponibilidad de los servidores

3. ***En relación con la computación en la nube (Cloud Computing)***
   - **A.** Idealmente, presenta la siguiente estructura en niveles: SaaS, PaaS, IaaS
   - **B.** Todas las demás afirmaciones son ciertas
   - **C.** Es posible gracias al uso de la tecnología de virtualización de equipos
   - **D.** Introduce un modelo de "pago por uso"

4. ***Suponemos definido el siguiente programa, llamado copy.js***

```
const fs=require('fs')
function copyFile(source, dest) {
  console.log("reading")
  fs.readFile(source, (error, data)=> {
    if (!error) {
    console.log("writing")
    fs.writeFile(dest, data, error=>{
      if (!error) console.log("done")
    })
    }
  })
}
copyFile("copy.js","copy2.js")
console.log("start")
```

Suponiendo que la lectura y escritura de ficheros no generan errores (existe el fichero a leer, tenemos los permisos necesarios, hay espacio para escribir el nuevo fichero, etc.), indica el resultado esperado al ejecutar el programa

- **A.** Error sintáctico al intentar ejecutar el programa
- **B.** Escribe en pantalla (en ese orden) las líneas

```
reading
start
writing
done
```

- **C.** Escribe en pantalla (en ese orden) las líneas

```
start
reading
writing
done
```

- **D.** Escribe en pantalla las líneas (`start`, `reading`, `writing`, `done`), pero no podemos saber el orden (puede ser un orden distinto en cada ejecución)

5. ***Suponemos definido el siguiente programa, llamado pcopy.js***

```
const fs=require('fs').promises
function copyFile(source, dest) {
  console.log("reading")
  fs.readFile(source)
  .then(data=>{console.log("writing");
         return fs.writeFile(dest, data)})
  .then(()=>console.log("done"))
}
copyFile("pcopy.js","pcopy2.js")
console.log("start")
```

Suponiendo que la lectura y escritura de ficheros no generan errores (existe el fichero a leer, tenemos los permisos necesarios, hay espacio para escribir el nuevo fichero, etc.), indica el resultado esperado al ejecutar el programa

A. Escribe en pantalla (en ese orden) las líneas

```
reading
start
writing
done
```

B. Escribe en pantalla (en ese orden) las líneas

```
start
reading
writing
done
```

C. Escribe en pantalla las líneas (`start`, `reading`, `writing`, `done`), pero no podemos saber el orden (puede ser un orden distinto en cada ejecución)

D. Error sintáctico al intentar ejecutar el programa

6. ***A continuación se presenta un programa cliente y un programa servidor:***

```
***cliente
let net=require('net');
let sock=net.connect({port:3000}, ()=>{
  for (let i=1; i<=10; i++) sock.write('hello');
});
sock.on('data', (x)=>{console.log(""+x)});

***servidor
const net=require('net');
const server=net.createServer( (sock)=>{
   sock.on('data', (x)=>sock.write('OK '+x));
}).listen (3000);
```

En relación con estos dos programas, seleccione la opción correcta:

A. El servidor termina al responder todos los mensajes que envía el cliente

B. Ni el programa cliente ni el programa servidor terminan

C. Ninguna de las demás afirmaciones es cierta

D. El cliente termina al recibir todos los mensajes que envía el servidor

7. ***Considérese el siguiente programa JavaScript:***

```
function f(x) {
  return function (y) {
    x++
    return x+y
  }
}
g=f(5, 10)
console.log(g(3))
```

Indique qué mostraría en pantalla ese programa durante su ejecución.

A. 14

B. 63

C. 9

D. Un error, por haber invocado a f() con más de un argumento

8. ***Considérese el siguiente programa JavaScript:***

```
setTimeout(()=>{console.log("1")},10)
let k=Math.abs(parseInt(process.argv[2]))
let j=0
for (let i=0; i<k; i++) j+=i
setTimeout(()=>{console.log("2")}, 1)
```

Indique qué mostraría en pantalla ese programa durante su ejecución si se le pasa un valor numérico entero como primer argumento en la línea de órdenes y ninguna de las operaciones aritméticas genera un error.

A. Los valores 1 y 2, cada uno en una línea, y su orden dependerá del valor facilitado en el argumento

B. Siempre una primera línea con un 1 y una segunda línea con un 2

C. Siempre una primera línea con un 2 y una segunda línea con un 1

D. Un error, pues no se puede utilizar más de una vez `setTimeout()` en un mismo programa

9. ***Considérese el siguiente programa JavaScript:***

```
const net=require('net')
let server=net.createServer(
  function(c){
    console.log('server connected')
    c.on('end',function(){
         console.log('server disconnected')
    })
    c.on('data', function(data){
         console.log('data from client: '
                     + data.toString())
         c.write(data)
    })
})// End of net.createServer()
server.listen(9000,
  function(){
    console.log('server bound')
})
```

Este programa muestra cómo podría escribirse un servidor muy sencillo que utilice un socket TCP. ¿Cómo consigue el proceso que ejecute este programa atender múltiples conexiones de sus clientes si solo dispone de un único hilo de ejecución para ello?

A. Permitiendo que cada conexión establecida sea gestionada por una llamada a la función definida como único argumento de `createServer()`

B. Gestionando las llegadas de mensajes en esas conexiones mediante eventos `"data"`

C. Utilizando la cola de eventos para secuenciar la ejecución de todos los "listeners" que se vayan activando durante el uso de esas conexiones

D. Todas las demás afirmaciones son ciertas

10. ***Considérese el siguiente programa JavaScript:***

```
const net=require('net')
let server=net.createServer(
  function(c){
    console.log('server connected')
    c.on('end',function(){
         console.log('server disconnected')
    })
    c.on('data', function(data){
         console.log('data from client: '
                     + data.toString())
         c.write(data)
    })
})// End of net.createServer()
server.listen(9000,
  function(){
    console.log('server bound')
})
```

Este programa ilustra cómo podría escribirse un servidor muy sencillo que utilice un socket TCP. ¿Qué mensaje mostrará en primer lugar (en caso de que no haya ningún error) un proceso que ejecute este programa?

A. `"server connected"`
B. `"server bound"`
C. O bien `"server bound"` o `"server connected"`
D. Cualquiera de entre los siguientes: `"data from client..."`, `"server connected"` o `"server bound"`

11. ***Considere el siguiente programa cliente que utiliza ZeroMQ para conectarse a dos servidores:***

```
const zmq=require('zeromq')
const rq=zmq.socket('req')
rq.connect('tcp://127.0.0.1:8888')
rq.connect('tcp://127.0.0.1:8889')
rq.send('Hello!')
rq.send('Hello again!')
rq.on('message', function (msg) {
  console.log('Response: '+msg)
})
```

Seleccione la afirmación correcta sobre el funcionamiento del programa:

A. Si no hay ningún servidor escuchando en el puerto 8889, este cliente no llega a mostrar nada en pantalla, pues su ejecución se bloqueará antes del `rq.on()`
B. El código de este cliente envía la cadena `"Hello!"` al servidor que gestione el puerto 8888 local y la cadena `"Hello again!"` al servidor que gestione el puerto 8889 local
C. Este programa no funcionará correctamente, pues necesitaría un segundo `rq.on('message'...)` para procesar la respuesta del segundo servidor
D. La ejecución del proceso que ejecute este programa finaliza tan pronto como se haya recibido la primera respuesta por parte de algún servidor

12. ***Los componentes de un sistema distribuido:***

A. Todas las demás afirmaciones son ciertas
B. Deben poder ser especificados y desarrollados de manera independiente
C. Deben poder ser desplegados autónomamente
D. Deben poder interactuar de manera sencilla y útil

13. ***Los estándares facilitan la superación de las complejidades, pero éstas NO incluyen***

A. La especificación de la funcionalidad de los servicios
B. El formato de la información transmitida
C. La sincronización entre cliente y servidor
D. La diferenciación de roles desempeñados en un sistema informático

14. ***En cuanto a los sistemas de mensajería:***
    - **A.** En las versiones persistentes el receptor dispone de un buffer para mensajes, que pueden enviarse incluso tras finalizar el emisor
    - **B.** Las versiones no persistentes con gestor no tienen sobrecarga por el uso de ese gestor
    - **C.** En las versiones no persistentes sin gestor el emisor y receptor deben mantener los mensajes en memoria
    - **D.** En las versiones no persistentes se exige que solo se pueda transmitir el mensaje si el receptor está activo

15. ***Usando ZeroMQ se ha desarrollado el código de dos componentes, que les permite comunicarse en dos ordenadores de una red IP. Van a realizarse cambios, y puede reutilizarse la mayoría del código anterior... :***
    - **A.** Si el cambio no supone que esos componentes pasen a ejecutarse en el mismo equipo
    - **B.** Ninguna de las demás afirmaciones es cierta
    - **C.** Solo si el lenguaje de programación de ambos componentes es el mismo
    - **D.** Si el cambio no supone que esos componentes pasen a ser hilos de un mismo proceso

16. ***Una de las ventajas del uso de ZeroMQ para programar componentes que se comunican es:***
    - **A.** Puedes elegir diferentes lenguajes para programar esos componentes
    - **B.** Ya no se necesitan sockets ni la semántica de las operaciones que los acompañan
    - **C.** Puedes comunicar componentes ZeroMQ con otros que usen el módulo `net` de NodeJS
    - **D.** Puedes comunicar componentes ZeroMQ con otros que usen el módulo `http` de NodeJS

17. ***Supongamos que en un equipo X disponemos de tres componentes A, B y C que emplean ZeroMQ. Indica qué afirmación es cierta:***
    - **A.** Si A y B son procesos que se comunican entre sí, se necesitan dos bibliotecas de ZeroMQ en el equipo X
    - **B.** Si A y B se comunican entre sí, y C lo hace con un componente de otro equipo Y, se necesitan dos bibliotecas de ZeroMQ en el equipo X
    - **C.** Si A, B y C son hilos de un mismo proceso que se comunican entre sí, se necesitan tres bibliotecas de ZeroMQ en el equipo X
    - **D.** Solo se necesita una biblioteca de ZeroMQ en el equipo X, independientemente del número de componentes que se comuniquen

18. ***Considere el siguiente programa JavaScript:***

```
for (let i=0; i<10; i++) {
  var j=i;
  setTimeout(()=>console.log("j = "+j), j*1000);
}
// Final del programa
```

En este código se programan 10 eventos, utilizando las variables `i` y `j`. Sobre estas variables podemos decir que:

- **A.** Al inicio del programa, ninguna variable está declarada
- **B.** Al final del programa, j tiene el valor 0, y se imprime la secuencia:`"j = 0"`, `"j = 1"`, `"j = 2"`... `"j = 9"` (en líneas separadas)
- **C.** Ninguna de las demás afirmaciones es cierta
- **D.** Al final del programa, ninguna variable está declarada (presente en el ámbito),y se imprime `"j = 9"` varias veces

19. ***Dado el siguiente código, elija la afirmación correcta:***

```
let fg
{
  let x=1
  fg= (y)=>{console.log("hello "+y+" "+ x++)}
}
for (let i=0; i<10; i++) fg("pp")
```

- **A.** El código es incorrecto pues la función `fg` no puede emplearse fuera de su bloque
- **B.** Al ejecutar el programa se imprimirán por la consola 10 líneas idénticas
- **C.** La variable `x` forma parte de una clausura
- **D.** No hay clausuras, pues no hay funciones que retornen funciones

20. ***Considera el siguiente programa, similar a los programas emitterX.js usados en el laboratorio 1:***

```
const ev=require('events')
const emitter=new ev.EventEmitter()
const e1='e1', e2='e2', e3='e3'
let inc=1

function handler(e,n) {
  return (inc)=> {
    n+=inc
    console.log(e + ' --> ' + n)
  }
}

emitter.on(e1, handler(e1,1))
emitter.on(e2, handler(e2,'2'))
emitter.on(e3, handler(e3,3))
emitter.on(e3, handler(e3,'3'))

function phase() {
  emitter.emit(e1,inc)
  emitter.emit(e2,0)
  inc++
}

setInterval(phase,1000)
setTimeout(process.exit,2500)
```

¿Cuál es la salida en pantalla al ejecutar el programa?

A.
```
e1 --> 2
e2 --> 20
e3 --> 4
e3 --> 31
e1 --> 4
e2 --> 200
e3 --> 6
e3 --> 312
```

B.
```
e1 --> 2
e2 --> 20
e1 --> 4
e2 --> 200
```

C.
```
e1 --> 2
e2 --> 21
e1 --> 4
e2 --> 212
```

D.
```
e1 --> 2
e2 --> 21
e3 --> 4
e3 --> 31
e1 --> 4
e2 --> 212
e3 --> 6
e3 --> 312
```

21. ***En la parte final de la práctica 1 se formula una cuestión en la que, dado un pool de servidores, el proxy debe enviar la petición al servidor menos cargado de dicho pool.***

A. Puede conseguirse modificando únicamente el código de programador.js

B. Puede conseguirse sin añadir ni modificar código en los equipos servidores

C. Puede conseguirse modificando únicamente el código de `ProxyProg.js` (o `ProxyConf.js`)

D. Se necesitará añadir (o modificar) código en los equipos servidores

22. ***Para implementar el proxy programable de la práctica 1...***

A. Ya no es necesario crear un socket específico para dialogar con el servidor (`serverSocket`)

B. Utilizamos `JSON.stringify` y `JSON.parse` para codificar/decodificar los mensajes entre el cliente y el proxy

C. En el proxy debemos invocar dos veces `net.createServer`: una para definir el código que atiende al cliente, y otra para definir el código que atiende al servidor

D. En el proxy debemos invocar dos veces `net.createServer`: una para definir el código que atiende al cliente, y otra para definir el código que atiende al programador