TSR - Rec_Primer Parcial. 2025-01-30

Este examen consta de 17 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 la peor respuesta, cada acierto suma 10/16 puntos y cada error descuenta 10/48 puntos. Debes contestar en la hoja de respuestas.

A

1. ***Esta aplicación puede considerarse un servicio distribuido cuando se despliegue en el entorno mencionado:***
   - **A.** Un compilador de Java al ser utilizado en todos los ordenadores de un mismo laboratorio para resolver una misma práctica de alguna asignatura de programación
   - **B.** Microsoft Word en Office 365 cuando es utilizada por múltiples usuarios en sus respectivos ordenadores para editar un mismo documento compartido entre todos ellos
   - **C.** El intérprete de órdenes bash al ser utilizado por un usuario en su ordenador para lanzar aplicaciones localmente, sin utilizar la red en ellas
   - **D.** Todas las afirmaciones son ciertas

2. ***Al estudiar la evolución de los servicios software se llega a identificar una etapa en la que su modelo de servicio consigue automatizar la monitorización de parámetros relevantes, permite expresar puntos de elasticidad y automatiza la reconfiguración del servicio en función de la carga existente. Ese modelo de servicio es:***
   - **A.** SaaS
   - **B.** PaaS
   - **C.** Cluster altamente disponible
   - **D.** IaaS

3. ***Al analizar los paradigmas de programación se indica que para que un servicio sea escalable, sus servidores no deberían suspenderse al gestionar cada petición. Eso puede proporcionarse adoptando este paradigma:***
   - **A.** Multi-hilo, pues aunque sus hilos puedan compartir recursos, eso no conducirá al bloqueo de ninguna actividad
   - **B.** Dirigido por eventos, asociando un hilo a cada evento y compartiendo memoria entre todos ellos
   - **C.** Multi-hilo, pues los múltiples hilos que se pudieran generar jamás compartirán recursos
   - **D.** Asincrónico, o dirigido por eventos, pues los nuevos eventos generados pueden mantenerse en una cola, sin interrumpir o bloquear la actividad en curso

4. ***¿Qué se muestra en pantalla al ejecutar el siguiente programa?***

```
const k=2
if (k==1)
  console.log("k=1")
else {
  var j=3
  console.log("j-k=" + (j-k))
}
console.log("j="+j)
```

   - **A.** Este contenido: `j-k=1, ...Reference error: j is not defined...`
   - **B.** Un error indicando que no se puede restar una constante a una variable
   - **C.** Dos líneas con: `j-k=1, j=3`
   - **D.** Dos líneas con: `k=1, j=undefined`

5. ***¿Qué se muestra en pantalla al ejecutar el siguiente programa?***

```
const k=2
if (k==1)
  console.log("k=1")
else {
  let j=3
  console.log("j-k=" + (j-k))
}
console.log("j="+j)
```

A. Un error indicando que no se puede restar una constante a una variable
B. Dos líneas con: `j-k=1, j=3`
C. Dos líneas con: `k=1, j=undefined`
D. Este contenido: `j-k=1, ...Reference error: j is not defined...`

6. ***¿Qué se muestra en pantalla al ejecutar el siguiente programa?***

```
function greet( x="John", y ) {
  console.log("Hi, "+x); console.log("Hi, "+y)
}
greet(undefined, "Mary", "Peter")
```

A. `Hi, Mary` `Hi, Peter`
B. `Hi,` `Hi, Mary`
C. `Hi, John` `Hi, Mary`
D. Un error, pues la función `greet` espera dos argumentos y le estamos pasando tres

7. ***¿Qué mensaje muestra en primer lugar este programa, cuántos callbacks se utilizan en él y cuántas clausuras?***

```
function generateF(x) {
  return function () {
    console.log("Writing after "+x+" seconds.")
  }
}
setTimeout(generateF(0), 0)
console.log("End!")
```

A. `Writing after 0 seconds`, con un callback y una clausura
B. `End!`, con un callback y una clausura
C. `Writing after 0 seconds`, sin callbacks y sin clausuras
D. `Writing after 0 seconds`, sin callbacks y con una clausura

8. ***Considere que este programa se inicia en un ordenador en el que ya se ejecuta un servidor net que atiende conexiones en el puerto 9000 y que este último siempre devuelve alguna respuesta a cualquier solicitud recibida. Seleccione la afirmación cierta sobre este programa cliente:***

```
const net=require('net')
let counter = 0
let client = net.connect({port:9000},
  function () {
    console.log("client connected!")
    client.write(counter+' world')
  })
client.on('data', function (data) {
  console.log(data.toString())
  if (counter == 9) client.end()
  else client.write((++counter) +' world')
})
client.on('end', function () {
  console.log("client disconnected")
})
```

A. El código utilizado en este cliente para enviar sus peticiones garantiza persistencia débil en la comunicación resultante
B. Este cliente finaliza tras recibir la respuesta a su décima solicitud
C. Si se sustituyera en la octava línea el primer argumento del método `on` por la cadena `message`, el programa seguiría funcionando de igual manera
D. Este cliente es incapaz de interactuar con algún servidor, pues no llega a enviarle ningún mensaje

9. ***Este es un ejemplo de middleware:***
A. Ubuntu 24.10
B. Node.js 22.12.0
C. LibreOffice Writer 24.8
D. Apache ActiveMQ Classic 6.1.4

10. ***Selecciona la afirmación correcta sobre los sistemas de comunicación no persistente:***
A. Si utilizan un gestor son más eficientes que cuando no lo necesitan
B. Si no utilizan un gestor son más eficientes que cuando lo necesitan
C. Suelen mantener los mensajes en colas del emisor cuando el receptor no esté disponible
D. Exigen que el receptor esté preparado para que el emisor transmita sus mensajes

11. ***Si `so` es un socket ZeroMQ de tipo REQ, entonces al realizar la operación `so.send("Ejemplo")`:***
    - **A.** Ese envío quedará bloqueado mientras no se reciba previamente alguna petición desde otro socket REP, pues REQ solo sirve para contestar y no puede iniciar la comunicación
    - **B.** Al transmitir el mensaje al receptor, tendrá dos segmentos: la identidad de `so` y `"Ejemplo"`
    - **C.** El mensaje `"Ejemplo"` permanecerá necesariamente en alguna cola de salida durante un largo intervalo, pues el patrón REQ-REP es sincrónico y bloqueante
    - **D.** Al transmitir el mensaje al receptor, tendrá dos segmentos: `""` y `"Ejemplo"`

12. ***Se pretende desarrollar, utilizando sockets ZeroMQ, un servicio de distribución de noticias formado por tres componentes:***
    (a) corresponsal, especializado en una determinada temática y que emitirá sus mensajes utilizando dos segmentos en ellos (temática y texto de la noticia),
    (b)agencia, que recibirá la información de los corresponsales y la reenviará a la cadena o cadenas interesadas y
    (c) cadena, que mediante un proceso periódico se conectará durante cierto intervalo a la agencia para recibir la información de interés para sus potenciales usuarios. Una misma cadena puede tener interés en diferentes temáticas.
    La comunicación será unidireccional: la agencia jamás responde al corresponsal y tampoco lo hace la cadena a la agencia. La información que envíe un corresponsal no debe perderse si la agencia todavía no ha iniciado su actividad. Por el contrario, la cadena solo está interesada en la información que la agencia publique desde el momento en que realice su conexión a la agencia, descartando los mensajes que la agencia haya publicado previamente.
    ¿Qué tipos de sockets podrían utilizarse para desarrollar este servicio?
    - **A.** Corresponsal: PUSH; Agencia: SUB y PUB; Cadena: PULL
    - **B.** Corresponsal: REQ; Agencia: REP y PULL; Cadena: PUSH
    - **C.** Corresponsal: PUB; Agencia: SUB y REQ; Cadena: REP
    - **D.** Corresponsal: PUSH; Agencia: PULL y PUB; Cadena: SUB

13. ***Considere el siguiente programa:***

```
const zmq = require("zeromq")
const sub = zmq.socket('sub')

sub.connect("tcp://localhost:5555")
sub.on("message", function(msg) {
  console.log("Received: " + msg)
})
```

    Se ha iniciado un proceso suscriptor que ejecuta este programa y en ese mismo ordenador se ha iniciado previamente un proceso que utiliza un socket PUB `p`, sobre el que se ha realizado con éxito la operación `p.bind("tcp://*:5555")`. Ese proceso publicador emite un mensaje cada segundo, de manera ininterrumpida, cuyo contenido son noticias breves de texto, entre 50 y 80 caracteres. Sin embargo, el proceso suscriptor no consigue recibir ni mostrar ningún mensaje. ¿Por qué motivo?
    - **A.** La gestión de las conexiones es errónea, porque los sockets SUB deben realizar `bind` y los PUB `connect`
    - **B.** El evento a utilizar para recibir los mensajes debe ser `"data"` en lugar de `"message"`
    - **C.** El programa publicador deber usar `"tcp://localhost:5555"` como argumento de la operación `bind`, en lugar de la URL que ha utilizado
    - **D.** El programa suscriptor debe incluir una instrucción `sub.subscribe("")` para recibir sus mensajes

14. ***Un proceso A emite empleando un socket PUSH, con un `bind("tcp://localhost:8888")` y `setInterval(...,1000)`, seis mensajes cuyo contenido es, respectivamente, `"1", "2", "3", "4", "5"` y `"6"`, antes de ser finalizado por el usuario. Tras haber sido iniciado y antes de que llegue a transcurrir el primer segundo, tres procesos B, C y D se han conectado, en ese orden, a ese puerto 8888 y reciben mensajes con un socket PULL. ¿Qué mensajes recibe cada proceso?***
    - **A.** B: "1" y "4"; C: "2" y "5"; D: "3" y "6"
    - **B.** B, C y D reciben, cada uno de ellos, todos los mensajes
    - **C.** Es imprevisible el conjunto de mensajes que recibirá cada uno
    - **D.** Ninguno, pues deberían haber utilizado otro tipo de socket para que la recepción fuera posible

15. ***Dado el siguiente programa:***

```
function f1 (a,b,c) {
  return a+b+c
}
let vector = [1,2,3,4]
console.log ("resultv1: " + f1 (...vector))
```

Elige el mensaje en pantalla que muestra su ejecución:

A. `resultv1: 1,2,3,4undefinedundefined`

B. `Uncaught ReferenceError: ...vector is not defined`

C. `resultv1: 6`

D. `resultv1: [1,2,3,4]`

16. ***En la segunda sesión de la práctica 1 se solicitaba extender un par de programas netClient.js y netServer.js para que el segundo devolviera al primero un valor proporcional a su carga en ese momento. Para ello, la versión extendida de netClient.js necesitaba recibir dos argumentos desde la línea de órdenes: la IP del servidor y la IP del cliente, en ese orden. ¿Con qué instrucciones podríamos acceder a esos argumentos?***

A. Todas las opciones son correctas

B.
```
let args = process.args.slice(2)
let ipServer = args[0]
let ipClient = args[1]
```

C.
```
let ipServer = process.argv[2]
let ipClient = process.argv[3]
```

D.
```
let pa = argv
let ipServer = pa[2]
let ipClient = pa[3]
```

17. ***En la tercera sesión de la práctica 1 se solicitaba revisar o implementar tres tipos de proxies: básico, configurable y programable. Cualquiera de ellos recibía los mensajes emitidos por un navegador web (o cualquier otro proceso cliente que utilizase conexiones TCP) para reenviarlos a un determinado servidor (normalmente, un sitio web). ¿Cuál es la principal diferencia entre el básico y el configurable?***

A. El básico solo permite gestionar un servidor, mientras que el configurable puede interactuar con dos servidores simultáneamente

B. El básico no gestiona ninguna caché, mientras que en el configurable podemos configurar el tamaño de la caché de páginas recientes que mantendrá

C. El código del básico mantiene la dirección y puerto del servidor en constantes, mientras que el configurable recibe esa información desde la línea de órdenes

D. El básico recibe la dirección y puerto del servidor desde la línea de órdenes, mientras que el configurable recibe esa información en un mensaje de configuración