TSR **Primer Parcial** **Temas 1,2,3 (1ª parte) Práctica 1** **2022-11-03** **A**

Este examen tiene un valor de 10 puntos, y consta de 24 cuestiones. Cada cuestión plantea 4 alternativas y tiene una única respuesta correcta. Cada respuesta correcta aporta 10/24 puntos, y cada error descuenta 10/72 puntos. Debe contestar en la hoja de respuestas.

1 *En la computación cooperativa:*

- **a** Las tareas de cómputo suelen realizarse en los *clientes,* mientras que el *servidor* reparte las tareas a realizar.
- **b** El fallo de un cliente impedirá que la computación global termine.
- **c** Los clientes suelen intercambiar resultados parciales entre ellos.
- **d** Todas las demás opciones son ciertas.

2 *Al analizar el ciclo de vida de los servicios, el rol* administrador de sistema *se centra en:*

- **a** Desarrollar las aplicaciones solicitadas por los usuarios.
- **b** Decidir qué componentes formarán parte de una aplicación distribuida que proporcione el servicio solicitado.
- **c** Decidir en qué nodos se instalará cada componente y con qué configuración, así como comprobar que cada ordenador funcione correctamente.
- **d** Ninguna de las demás opciones es cierta.

3 *Una ventaja del modelo de programación asincrónica cuando se compara con la programación concurrente basada en múltiples hilos de ejecución es:*

- **a** Se reduce en gran medida la probabilidad de que haya condiciones de carrera.
- **b** Mejora el rendimiento: no sólo lanza distintos hilos, sino que dichos hilos no pasan nunca a estado suspendido.
- **c** Por utilizar comunicación no persistente, los clientes no necesitan conectarse a ningún servidor antes de enviar sus peticiones.
- **d** Todas las demás opciones son válidas.

4 *En la Wikipedia, la replicación de componentes …:*

- **a** No resulta necesaria en ninguno de los componentes del sistema.
- **b** Suele utilizarse en sus componentes, combinada con otros mecanismos como pueda ser el uso de cachés y la distribución o sharding de datos.
- **c** Se utiliza únicamente en el componente *Apache.*
- **d** Se utiliza únicamente en el componente *MySQL.*

5 *Considere el siguiente programa JavaScript:*

```
const fs=require("fs")

console.log("Call to asynchronous readFile")
fs.readFile("/proc/loadavg",(e,d)=> {
   if (e) console.error(e.message)
   else console.log(d+"")
})
console.log("End of asynchronous readFile")

console.log("Call to synchronous readFile")
console.log( fs.readFileSync("/proc/loadavg") + "" )
console.log("End of synchronous readFile")
```

¿Cuál será la última cadena que este programa mostrará, si el fichero que se intenta leer existe, tiene contenido, y no hay errores en sus lecturas?

- **a** `End of asynchronous readFile`
- **b** El contenido del fichero `/proc/loadavg`
- **c** `End of synchronous readFile`
- **d** Ninguna de las demás opciones es correcta.

Considérese el siguiente conjunto de programas JavaScript:

```
// Program: ex1.js
const ev = require('events')
const emitter = new ev.EventEmitter()
const e1 = "print"
emitter.on(e1, function(num) {
  return () =>
    console.log("Event " + e1+ ": " + ++num)
}(0))
emitter.emit(e1)
```

```
// Program: ex2.js
function f(x) {
  return function (y) {
    let z = x + y
    return 20
  }
}
```

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

6 *¿Qué mostrará en pantalla la ejecución de* `ex1.js`*?*

- **a** Mostrará Event print: 1
- **b** Mostrará una sucesión de mensajes:
  Event print: 1
  Event print: 2
  Event print: 3
  …
- **c** Se generará una excepción al ejecutar la línea `emitter.emit(e1)`.
- **d** No llegará a mostrarse ningún mensaje.

7 *¿Cuántas funciones anónimas (no confundir con llamadas a funciones) se definen en* `ex1.js`*?*

- **a** Ninguna de las demás opciones es cierta.
- **b** Infinitas, pues se da una definición recursiva.
- **c** Una.
- **d** Dos.

8 *¿Qué valor devuelve una llamada a la función* `f` *con el argumento* `10`*, es decir,* `f(10)`*, en el programa* `ex2.js`*?*

- **a** undefined
- **b** 10
- **c** Una función.
- **d** NaN

9 *¿Podría el programa* `server.js` *mantener múltiples conexiones de procesos clientes?*

- **a** No podría, pues JavaScript solo tiene un hilo de ejecución y cada conexión con un cliente necesita ser gestionada por un hilo diferente.
- **b** Podría, pues tanto el establecimiento y el cierre de la conexión, como la llegada de nuevos mensajes o la ocurrencia de errores en la conexión se modelan como eventos y JavaScript dispone de una cola de eventos para gestionarlos, haya los que haya.
- **c** No podría, pues solo existe un callback para gestionar el establecimiento de una conexión y mientras éste se ejecute, no puede haber otra actividad en ejecución.
- **d** En principio podría, pero este programa no llega a hacerlo porque no dispone de ningún `listener` para el evento `connect`.

10 *Considere que se dispone de una función* `countLines(text)` *que devuelve el número de líneas contenidas en la cadena de texto facilitada en su parámetro* `text`. *Si se necesitara escribir un programa Node.js que utilizara promesas y mostrase el número de líneas contenidas en el fichero* `Ejemplo.txt` *del directorio actual (existente y con permiso de lectura), una posible solución sería:*

a Ninguna, pues con promesas no es posible realizar esa gestión.

b

```
const fs=require('fs')
function countLines(t) { return t.split('\n').length }
console.log(countLines(
  fs.readFileSync("Ejemplo.txt",'utf-8'))
```

c

```
const fs=require('fs').promises
function countLines(t) { return t.split('\n').length }
fs.readFile("Ejemplo.txt",'utf-8')
.then(countLines).then(console.log)
```

d

```
const fs=require('fs').promises
function countLines(t) { return t.split('\n').length }
fs.readFile("Ejemplo.txt",'utf-8').then(console.log)
```

11 *Indique la afirmación correcta respecto a Node.js:*

a Node está implementado internamente mediante un único hilo de ejecución. Es decir Node en sí mismo es mono-hilo.

b Node integra el motor de JavaScript V8, el mismo motor de JavaScript que utiliza Google Chrome.

c Node resulta adecuado para ejecutar aplicaciones realizadas en JavaScript y en Java.

d Node y toda la funcionalidad de sus módulos está integrada en los navegadores web modernos, como Google Chrome o Firefox.

12 *Si ejecutamos el siguiente fragmento de código indique la afirmación correcta:*

```
const ev = require ('events');
const emitter = new ev.EventEmitter
emitter.on ("event1", (x) => console.log (x));
emitter.emit ("event1", "hello");
emitter.emit ("hello");
console.log ("bye")
```

a Se imprimirá la cadena `hello` por la consola y después se imprimirá `bye`

b Producirá un error, pues se emite el evento `hello` y no hay ningún manejador para dicho evento.

c Se imprimirá la cadena `bye` por la consola y después se imprimira `hello` .

d Si ejecutamos el programa varias veces, unas veces se imprimirá `hello` antes que `bye` y otras veces se imprimirá `bye` antes que `hello` .

13 *Respecto al broker (o gestor) en los sistemas de mensajería, selecciona la afirmación verdadera:*

a Los sistemas sin broker no son adecuados cuando se desea persistencia débil

b Un sistema con broker puede ser construido a partir de otro sin broker

c Los sistemas con broker facilitan, aunque no proporcionan directamente, una visión de estado compartido

d Un sistema sin broker puede ser construido a partir de otro con broker

14 *ØMQ, como sistema de mensajería, intenta cubrir varios objetivos. Indica cuál NO lo es:*

a Facilidad en su uso para los programadores por reproducir una API similar a los sockets BSD

b Fiabilidad garantizada mediante el seguimiento automático de los mensajes pendientes de recepción

c Reutilización del mismo código, salvo la URL, para comunicar hilos, procesos en el mismo equipo, y procesos en equipos diferentes

d Facilidad para resolver problemas comunes al soportar patrones básicos de interacción

15 *Sea un emisor cuyo código conecta sin errores con receptores que han efectuado la operación bind correspondiente con un socket REP de ØMQ:*

```
const zmq = require('zeromq')
const q = zmq.socket('req')
q.connect('tcp://10.0.0.1:8887')
q.connect('tcp://10.0.0.2:8888')
q.connect('tcp://10.0.0.3:8889')
```

Si a continuación el emisor intenta ejecutar estas 3 instrucciones:

```
q.send('Hello A')
q.send('Hello B')
q.send('Hello C')
```

Ocurrirá esto:

- **a** Cada receptor recibe los 3 mensajes en el mismo orden en que se envían
- **b** El emisor no puede iniciar la invocación del segundo q.send hasta recibir respuesta del primero
- **c** Cada receptor recibe solo uno de los mensajes, pero el socket emisor debe esperar respuesta para cada uno antes de propagar el siguiente
- **d** Cada receptor recibe solo uno de los mensajes, y el socket emisor solo debe esperar respuesta tras propagar el último de los tres

16 *¿En qué condiciones varios sockets ØMQ de tipo REQ se pueden interconectar?*

- **a** Solo cuando haya exactamente uno realizando la operación bind
- **b** Solo cuando no haya más de un envío simultáneo por el socket
- **c** Solo cuando se emplee transporte local (IPC o hilos)
- **d** No se pueden interconectar porque REQ necesita que en el otro extremo se utilice un socket complementario

17 *Se ha desarrollado un programa servidor que utiliza un socket REP para interactuar con los clientes, que utilizan sockets REQ. Sin embargo, esta solución solo permite procesar una solicitud en cada momento y convendría mejorar su rendimiento. Para ello, un programador propone que tanto los clientes como el servidor utilicen un único socket PUSH para enviar mensajes y un único socket PULL para recibirlos. Si habitualmente habrá múltiples clientes conectados e interactuando con el servidor, ¿será esta una solución correcta?*

- **a** No, porque el socket PUSH no puede ser utilizado para enviar mensajes: solo permite recibirlos.
- **b** No, porque el socket PUSH envía los mensajes siguiendo el orden rotatorio de conexión y podría enviar respuestas a clientes que no han enviado ninguna petición.
- **c** Sí, porque el patrón PUSH/PULL no bloquea en ningún caso el envío o recepción de mensajes y el rendimiento mejorará claramente.
- **d** Ninguna de las demás afirmaciones es cierta.

18 *En 0MQ, en el establecimiento de una conexión:*

- **a** Cada socket solo puede hacer, a lo más, una operación bind.
- **b** Cada socket solo puede hacer, a lo más, una operación connect.
- **c** La operación bind siempre debe preceder a las operaciones connect.
- **d** Las restantes respuestas son erróneas.

19 *Entre las fuentes de complejidad de un sistema distribuido se encuentran:*

- **a** La solicitud de servicios.
- **b** El formato de la información.
- **c** La gestión de fallos.
- **d** Las restantes respuestas son válidas.

20 *Indique el resultado que se verá por la consola al ejecutar el siguiente fragmento de código:*

```
function g (a) {
  a = a + 1
  return (b) => { return b + a }
}
let f = g(1)
console.log ( f(1) + g (1)(1) )
```

a undefined

b 6

c 7

d NaN

21 *Toma el siguiente fragmento de código e indica, en los primeros 4 segundos (inclusive), cuántas veces se mostrará el texto* `Evento uno tratado` *por pantalla.*

```
const ev = require('events')
const emitter = new ev.EventEmitter()
const t = 1000, e1 = "uno"

var handler = function () {
   console.log("Evento "+e1+" tratado")
}

emitter.on(e1, handler)

function etapa() {
  emitter.emit(e1)
  setInterval(etapa, t)
}

etapa()
```

a Ninguna porque hay un error en la función proporcionada como argumento de emitter.on

b Ninguna porque hay un error en la función proporcionada como argumento de setInterval

c 4 ó 5

d Al menos 7

22 *Dado el siguiente programa, indique la afirmación correcta:*

```
let MAX=1000

console.log ("uno")
setTimeout ( () => console.log ("dos"), 100)

// Hacer trabajo
for (let i=0; i<MAX; i++) {}

setTimeout ( () => console.log ("tres"), 99)
setTimeout ( () => console.log ("cuatro"), 0)
console.log ("cinco")
```

a Cuando lo ejecutemos, siempre veremos `cinco` antes que `cuatro` .

b Cuando lo ejecutemos, es posible que veamos `cuatro` antes que `cinco` .

c Cuando lo ejecutemos, siempre veremos `dos` antes que `tres`

d El programa imprime `uno` y `cinco` y termina sin imprimir nada más.

23 *Consideremos el siguiente fragmento del código del proxy inverso visto en la tercera sesión de la práctica 1:*

```
const net = require('net')
const ...

const server = net.createServer(function (s2) {
    const s1 = new net.Socket()
    s1.connect(parseInt(REMOTE_PORT),
REMOTE_IP, function () {
     ***
    })
}).listen(LOCAL_PORT, LOCAL_IP)
console.log("TCP server accepting connection" +
  "on port: " + LOCAL_PORT)
```

*Las instrucciones que faltan en las líneas *** son:*

a s2.on('data', function (m2) {s1.write(m2)})
s1.on('data', function (m1) {s2.write(m1)})

b s1.on('data', function (m1) {s2.write(m1)})
s2.on('data', function (m2) {s1.write(m2)})

c Ninguna de las demás opciones es válida.

d Las dos opciones que muestran código son válidas.

24 *El ejercicio del proxy programable incluye un nuevo servicio del proxy que, resumidamente, permite modificar REMOTE_PORT y REMOTE_IP. Imagina que para ello se propone incorporar el siguiente código al proxy programable:*

```
// s1 conecta al proxy con el cliente.
// s2 conecta al proxy con el servidor remoto.
const prog = net.createServer((s3)=>{
  s3.on('data', function(m3) {
  ***
  })
})
```

Y que la petición enviada por programador.js se construye a partir de argv de la siguiente forma:

```
s.write(JSON.stringify({
   "remote_ip": argv[2],
   "remote_port": parseInt(argv[3])
}))
```

*Las instrucciones que faltan en las líneas *** del proxy programable propuesto son:*

a

```
s2.end();
REMOTE_IP = JSON.parse(m3).remote_ip;
REMOTE_PORT = JSON.parse(m3).remote_port
```

b

```
s1.end();
REMOTE_IP = JSON.parse(m3).remote_ip;
REMOTE_PORT = JSON.parse(m3).remote_port
```

c

```
REMOTE_IP = JSON.parse(m3).remote_ip;
REMOTE_PORT = JSON.parse(m3).remote_port
```

d

```
REMOTE_IP = m3.remote_ip;
REMOTE_PORT = m3.remote_port
```

TSR **Primer Parcial** **3 Noviembre 2022** **A**

Rellena y entrega la siguiente hoja de respuestas. Cada cuestión posee una única respuesta correcta. No olvides cumplimentar correctamente tus datos personales.

No taches una posible respuesta incorrecta: bórrala o cúbrela con Tipp-Ex

Una cuestión con más de una respuesta marcada se considera no contestada

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9

DNI: _____

Apellidos: _____

Nombre: _____

1	A	B	C	D	**21**	A	B	C	D
2	A	B	C	D	**22**	A	B	C	D
3	A	B	C	D	**23**	A	B	C	D
4	A	B	C	D	**24**	A	B	C	D
5	A	B	C	D					
6	A	B	C	D					
7	A	B	C	D					
8	A	B	C	D					
9	A	B	C	D					
10	A	B	C	D					
11	A	B	C	D					
12	A	B	C	D					
13	A	B	C	D					
14	A	B	C	D					
15	A	B	C	D					
16	A	B	C	D					
17	A	B	C	D					
18	A	B	C	D					
19	A	B	C	D					
20	A	B	C	D					