NIST	First Partial	Units 1, 2 and 3. Lab 1	2021-10-28	A

This exam has a maximum mark of 10 points and consists of 32 questions. Each question has 4 choices and only one of them is correct. Each correct answer provides 10/32 points and each error discounts 10/96 points. You should deliver the answer sheet.

1 *Entre los mecanismos utilizados en la Wikipedia para mejorar las prestaciones destacan:*

- **a** Replicación.
- **b** Equilibrado de carga y cachés mediante el uso de proxies inversos.
- **c** Todos ellos.
- **d** Particionado de los datos persistentes.

2 *El modelo de computación teórico basado en guardas y acciones:*

- **a** Se corresponde directamente con la programación orientada a eventos.
- **b** Solo se aplica a sistemas de mensajería.
- **c** Está exclusivamente relacionado con la programación multihilo.
- **d** Únicamente tiene sentido en aplicaciones con un único proceso.

3 *Entre los aspectos relevantes de los sistemas distribuidos destacan:*

- **a** La tolerancia a fallos.
- **b** El uso compartido de determinados recursos.
- **c** Todos ellos.
- **d** Coordinar las acciones de diferentes componentes.

4 *¿Cuál de los siguientes elementos relacionarías con el término LAMP?*

- **a** Un middleware de mensajería.
- **b** Un tipo de contenedores de aplicaciones.
- **c** Un módulo de Node.js.
- **d** La Wikipedia.

5 *Sobre la computación cooperativa:*

- **a** La Wikipedia es un ejemplo.
- **b** Tiene todas estas características.
- **c** Se utiliza fundamentalmente en sistemas Peer-to-Peer.
- **d** Es adecuada para resolver problemas científicos e ingenieriles que puedan particionarse en otros más sencillos.

6 *En la computación en la nube:*

- **a** Los proveedores SaaS proporcionan al usuario servicios de software.
- **b** Los proveedores IaaS proporcionan máquinas virtuales para ejecutar los servivios requeridos.
- **c** Los proveedores PaaS proporcionan un entorno de desarrollo con características de sistema operativo.
- **d** Todas las respuestas son válidas.

7 *Considere el programa* `First fragment` *basado en el módulo* `events` *de Node.js. Al ejecutar ese código, se muestra en pantalla:*
`Event print: 1`

```
// First fragment
const ev = require('events')
const emitter = new ev.EventEmitter()
const e1 = "print"
let n = 0     // Event counter
emitter.on(e1, function() {
  console.log("Event " + e1 + ": " + ++n)
})
emitter.emit(e1)
```

¿Qué se mostrará en pantalla si modificamos el programa y lo dejamos como en Second fragment (donde se ha desplazado únicamente la última línea del programa original)?

```
// Second fragment
const ev = require('events')
const emitter = new ev.EventEmitter()
const e1 = "print"
let n = 0     // Event counter
emitter.emit(e1) // emit() before on()
emitter.on(e1, function() {
  console.log("Event " + e1 + ": " + ++n)
})
```

a Ninguna de las demás opciones es correcta.

b Se mostrará lo mismo que en la versión anterior: `Event print: 1`

c No llegará a mostrarse ningún mensaje.

d Se generará una excepción al ejecutar la línea emitter.emit(e1).

8 *Los siguientes programas utilizan el módulo* `net` *de Node.js. ¿Cuántas conexiones clientes concurrentes podría gestionar server.js?*

```
// File: server.js
const net = require('net')
let server = net.createServer(
  function(c) { //'connection' listener
   console.log('server connected')
   c.on('end', function() {
     console.log('server disconnected')
   });
   // Handle requests
   c.on('data', (msg) => {
     c.write(parseInt(msg) * 2 + "")
   })
}); // End of net.createServer()
server.listen(9000,
  function() { //'listening' listener
   console.log('server bound')
  });
```

```
// File: client.js
const net = require('net')
// The server is in our same machine.
let client = net.connect({port: 9000},
  function() { //'connect' listener
   console.log('client connected')
   client.write(process.pid + "") // Send my PID
  });
client.on('data', function(data) {
  // Write the received data to stdout.
  console.log(data.toString());
  client.write(parseInt(data)++ + "")
});
client.on('end', function() {
  console.log('client disconnected')
});
```

a Una. La gestión no sería concurrente: solo podría atenderse una nueva conexión cliente cuando la previa se hubiera cerrado.

b Dos.

c Ninguna.

d Su número no está, a priori, limitado.

9 *Suponga que ejecutamos el siguiente código en NodeJS:*

```
function f (cb) { cb() }
f( () => console.log ("hola") )
```

a La función f está declara de forma incorrecta, pues el argumento cb no tiene ningún tipo definido.

b El programa se ejecutará completamente en un único turno del bucle de eventos.

c Ninguna de las demás afirmaciones es cierta.

d La segunda línea invoca a la función f, con un callback que se ejecutará de forma asíncrona.

10 *Sobre el módulo* `net` *de NodeJS*

a Proporciona sockets TCP, mediante los que podremos enviar y recibir datos por la red. Para enviar datos empleamos el método write() y para recibir datos, empleamos el método read(). El método write(), retorna el número de bytes transferidos en la escritura y el método read() retorna un vector de bytes con los datos leídos.

b ninguna de las demás afirmaciones es cierta

c Para crear un socket, debemos especificar el tipo de socket que deseamos crear, que podrá ser entre otros 'rep' o 'req'

d Dado que los sockets TCP que proporciona el módulo 'net' son instancias de EventEmitter, podemos hacer que un proceso ejecute 'socket.emit(evento)', y este evento lo podremos recibir en el otro extremo de la conexión utilizando el método 'socket.on(evento, callback)'

11 *Dado el siguiente programa NodeJS.*

```
const events = require('events');
const ev = new events.EventEmitter();
function genera_cb () {
  let cs = [0,0];
  return ev => {
    console.log ("e" + ev + ":" + ++cs[ev]);
  }
}
ev.on ("e0", genera_cb() )
ev.on ("e1", genera_cb() )
ev.emit ("e0", 0)
ev.emit ("e0", 0)
ev.emit ("e1", 1)
```

a Al ejecutarlo veremos por la consola varias líneas, siempre las mismas, pero el orden de estas líneas podrá ser diferente de una ejecución a otra, pues se trata de un programa asíncrono.

b El programa contiene algún error que impide su ejecución. Concretamente, la llamada a la función 'genera_cb()" no se puede utilizar como manejador de los eventos.

c Si ejecutamos el programa varias veces, alguna de ellas podría producir la siguiente salida:
e0:1
e1:1
e0:2

d Al ejecutarlo, obtendremos por la consola siempre el resultando:
e0:1
e0:2
e1:1

12 *¿Cuántos parámetros debe usar la rutina o función que gestiona la ocurrencia de un evento?*

a Dos: el nombre del evento y el identificador de la propia rutina gestora.

b Uno: el nombre del evento.

c Ese número depende del número de argumentos utilizados en la operación emit() correspondiente.

d Ninguno. Esos callbacks no pueden tener ningún parámetro.

13 *Indica el resultado al ejecutar el siguiente fragmento de código:*

```
function f (n) {return n<2? 1:f (n–2)+f (n–1)}
setTimeout(()=>console.log(1),1000)
f (36) // requiere mas de 3000ms
console.log(2)
setTimeout(()=>console.log(3),100)
```

a Escribe en pantalla 2 3 1

b Escribe en pantalla 1 2 3

c Escribe en pantalla 2 1 3

d Escribe en pantalla 1 3 2

14 *Indica el resultado que se escribe en pantalla al evaluar el siguiente fragmento de código*

```
const f = function () {
for (var i=0; i<3; i++)
  setTimeout(() => console.log(i), (3–i)*1000)
}
f()
```

a Escribe (en líneas consecutivas) 3, 3, 3

b Escribe (en líneas consecutivas) 0, 1, 2

c Genera un error durante la ejecución

d Escribe (en líneas consecutivas) 2, 1, 0

15 *Ejecutamos el siguiente programa JS y observamos los dos resultados mostrados por pantalla*

```
function fuera(x, y) {
  return (z) => {return x[y](z+0)}
}
var v = [(a)=>{return a+1}, (a)=>{return a+3},
(a)=>{return a+2}]
let w = fuera(v, 1)
console.log (w(v[2]("cero")))
console.log (w(v[0](1)))
```

Esos resultados cumplen:

a Genera un error pero también muestra un número que comienza por 5

b Escribe 7 caracteres en la primera línea y un número terminado en 5 en la segunda

c Genera un error tras mostrar 7 caracteres en una línea

d Genera un error y no muestra ningún otro resultado

16 *Suponga que tenemos los siguientes programas cliente.js y servidor.js. Lanzaremos únicamente una instancia de cada uno.*

```
// cliente.js
let req = require('zeromq').socket ("req")
req.bind ("tcp://*:9999")
req.send ("hola")
req.on ("message", msg => {console.log (msg + "");
req.close()})
```

```
// servidor.js
let rep = require('zeromq').socket ("rep")
rep.connect ("tcp://localhost:9999")
rep.on ("message", msg =>
  rep.send (msg.toString().toUpperCase()));
```

a No importa en qué orden lancemos el cliente y el servidor. En ambos casos el resultado será el mismo. Veremos por la consola del cliente la cadena HOLA y el cliente terminará.

b Si lanzamos el servidor una vez, y después lanzamos el cliente, veremos la cadena HOLA por la consola, tras lo cual el cliente terminará. Si después de que termine el cliente, lanzamos otra vez el cliente, esta segunda ejecución no producirá ninguna salida por la consola.

c El código no es correcto, pues 'cliente' llama a bind(), mientras 'servidor' llama a connect(). Debe hacerse al revés.

d Si primero ejecutamos el 'servidor' y después ejecutamos el 'cliente', veremos por la consola del 'cliente' la cadena HOLA, tras lo cual el 'cliente' terminará. El 'cliente' termina debido a que cierra su socket, lo que provocará que también termine el programa 'servidor'.

17 *Entre la afirmaciones FALSA sobre ZeroMQ*

a ZeroMQ puede utilizar como transporte TCP o IPC (interprocess communication)

b ZeroMQ proporciona persistencia fuerte. Los mensajes enviados por cierto emisor, se almacenan en disco (almacenamiento estable), hasta que el receptor de los mensajes los reciba.

c Los sockets REP/REQ son sockets asíncronos, con los que se implementa el patrón sincrónico de petición/respuesta.

d ZeroMQ implementa 'framing', aspecto no contemplado en TCP/IP; p.ej. el módulo 'net' de NodeJS no proporciona 'framing'.

18 *Supongamos definidos los siguientes programas*

```
// client.js
// uso: node client id numPorts port1 port2 ... msg
msg ...
const zmq = require('zeromq')
const args= process.argv.slice(2)
const id=args[0], numPorts=parseInt(args[1])
const port=args.slice(2,2+numPorts)
const msg=args.slice(2+numPorts)
const req = zmq.socket('req')
for (let i=0; i<numPorts; i++)
  req.connect('tcp://127.0.0.1:'+port[i])
for (let i=0; i<msg.length; i++) {
  ... // otras acciones
  req.send([id,msg[i]])
}
req.on('message',
(c,id,m)=>console.log(c+':'+id+':'+m))
```

```
// server.js
// uso: node server id port
const zmq = require('zeromq')
const args= process.argv.slice(2)
const id=args[0], port=args[1]
const rep = zmq.socket('rep')
rep.bind('tcp://*:'+port)
rep.on('message', (c,m)=>{
  ... // otras acciones
  rep.send([c,id,m])
}
```

Si lanzamos simultáneamente las siguientes órdenes en terminales distintos:

```
node client C1 1 8000 A B C
node client C2 1 8000 D E
node server S1 8000
```

a Ninguna de las demás opciones es correcta.

b Con independencia de lo que cueste completar las otras acciones (…) en cliente y servidor, en la cola de salida del req de C1 no puede haber más de un mensaje pendiente.

c Con independencia de lo que cueste completar las otras acciones (…) en cliente y servidor, en la cola de entrada del rep de S1 puede haber más de un mensaje pendiente.

d En la cola de entrada del rep de S1 pueden haber simultáneamente peticiones de C1 y de C2.

19 *Supongamos definidos los siguientes programas*

```
// client.js
// uso: node client id numPorts port1 port2 ... msg
msg ...
const zmq = require('zeromq')
const args= process.argv.slice(2)
const id=args[0], numPorts=parseInt(args[1])
const port=args.slice(2,2+numPorts)
const msg=args.slice(2+numPorts)
const req = zmq.socket('req')
for (let i=0; i<numPorts; i++)
  req.connect('tcp://127.0.0.1:'+port[i])
for (let i=0; i<msg.length; i++) {
  ... // otras acciones
  req.send([id,msg[i]])
}
req.on('message',
(c,id,m)=>console.log(c+':'+id+':'+m))
```

```
// server.js
// uso: node server id port
const zmq = require('zeromq')
const args= process.argv.slice(2)
const id=args[0], port=args[1]
const rep = zmq.socket('rep')
rep.bind('tcp://*:'+port)
rep.on('message', (c,m)=>{
  ... // otras acciones
  rep.send([c,id,m])
}
```

Si lanzamos simultáneamente las siguientes órdenes en terminales distintos:

```
node client C 2 8000 8001 A B C D E
```

```
node server S1 8000
```

```
node server S2 8001
```

a El código anterior no se ejecuta correctamente, ya que no es posible conectar un cliente a varios servidores.

b Con independencia de lo que cueste completar las otras acciones (…) en cliente y servidor, los mensajes se procesan en orden A,B,C,D,E.

c El orden en que se procesan las peticiones depende de las duración de las otras acciones (…) en cliente y servidor.

d Con independencia de lo que cueste completar las otras acciones (…) en cliente y servidor, los mensajes se procesan en orden A,D,B,E,C.

20 *Indique la afirmación correcta*

a Un mismo nodo puede definir un socket de tipo pull y otro de tipo push.

b Un socket de tipo pull no puede tener conectado más de un socket de tipo push.

c Un socket de tipo push no posee cola de salida.

d Un socket de tipo push no puede tener conectado más de un socket de tipo pull.

21 *Indique la afirmación correcta sobre ZeroMQ*

a Los mensajes que envía un socket de tipo pub se reparten entre los suscriptores según una política Round Robin (RR).

b Un suscriptor no puede seleccionar el tipo de mensajes que recibe del publicador.

c La opción subscribe en un socket de tipo pub permite indicar a qué socket pub se conecta.

d Un suscriptor que se lanza una vez que el publicador ya está en marcha podría perderse parte de los mensajes.

22 *En relación con las operaciones bind y connect de 0MQ, indique la afirmación correcta*

a Todas las restantes opciones son falsas.

b No pueden realizarse varias operaciones connect sobre una URL donde únicamente se ha hecho una operación bind.

c Si se intenta la operación connect y la otra parte no ha realizado la operación bind, el programa aborta.

d Si se intenta la operación bind y la otra parte no ha realizado la operación connect, el programa aborta.

23 *¿Podemos difundir mensajes desde un mismo proceso emisor a dos procesos receptores sin usar PUB/SUB?*

a Sí: en lugar de pub usamos un vector con un req por suscriptor, y en lugar de sub usamos rep. Las operaciones pub.send(m) se sustituyen por req[i].send(m) sobre cada uno de los sockets del vector req.

b Sí: en lugar de pub usamos un vector con un push por suscriptor, y en lugar de sub usamos pull. Las operaciones pub.send(m) se sustituyen por push[i].send(m) sobre cada uno de los sockets del vector push.

c Sí: en lugar de pub usamos push, y en lugar de sub usamos pull. Las operaciones pub.send(m) se sustituyen por push.send(m).

d No.

24 *Sobre la posibilidad de que un subscriptor reciba únicamente parte de los mensajes publicados (filtrado):*

a No es posible. Siempre se reciben todos.

b El publicador puede indicar mediante subscribe() el tipo de mensajes a los que se suscribe cada suscriptor.

c El suscriptor puede indicar mediante subscribe() el tipo de mensajes a los que se suscribe.

d El suscriptor puede indicar el tipo de mensajes a los que se suscribe conectando con la URL que se pasa a la operación sub.connect.

25 *Indica qué salida escribe este programa por pantalla (cada dígito de las soluciones propuestas en una línea distinta):*

```
function c() {
  var x=0
  return ()=>{console.log(x); x++}
}
setInterval(c(), 1000)
setTimeout (c(), 2500)
setTimeout (()=>process.exit(0), 3500)
```

a 0 1 0 2.

b nada (genera un error).

c 0 0 1 2 3.

d secuencia creciente 0 1 2 … que no acaba hasta abortar el programa con ctrl-c.

26 *Indica qué salida escribe este programa por pantalla (cada dígito de las soluciones propuestas en una línea distinta):*

```
function f(x) {
  var y=0
  if (x>0) {
    let x=2
    console.log(x)
    y++
  }
  console.log(x+y)
}
f(8)
f(6)
```

a 2 9 2 7.

b 2 3 2 3.

c nada (genera un error).

d 2 9 2 8.

27 *Indica qué salida escribe este programa por pantalla:*

```
var a=[1,2,3,4], b = [1,2,3,4], c=a
console.log([a==b, a==c, a===b, a===c])
```

a ninguno (el programa genera un error).

b [false, false, false, false].

c [true, true, true, true].

d [false, true, false, true].

28 *Para el problema del emisor3, se pide utilizar setTimeout() para iniciar la llamada a la primera etapa. Indique entre las siguientes afirmaciones, aquella que sea cierta.*

a La función etapa() no debe llamar de nuevo a setTimeout(), pues la llamada a setTimeout() ya ha sido realizada. Es mejor llamar recursivamente a etapa(), sin llamar de nuevo a setTimeout().

b Ninguna de las demás afirmaciones es correcta.

c Podríamos sustituir setTimeout() por setInterval(). Dado que setInterval() programa el temporizador de forma periódica, no sería necesario llamar de nuevo a setInterval() ni a setTimeout() dentro de la función etapa().

d Llamar a setTimeout() dentro de la función etapa() implica hacer una llamada recursiva que puede agotar la pila del proceso.

29 *Sobre el problema de netServerLoad y netClientLoad, visto en prácticas. Sabiendo que la función getLoad() que se proporciona ya implementada, consulta la carga del equipo, indique la afirmación correcta.*

a El programa netClientLoad, debe contener en su código la función getLoad().

b Ninguno de los dos programas debe contener la función getLoad().

c Ambos programas, tanto netClientLoad como netServerLoad, deben contener en su código la función getLoad().

d El programa netServerLoad, debe contener en su código la función getLoad().

30 *El proxy programable (ProxyProg) solo sirve cuando el cliente emite órdenes HTTP porque …*

a Es muy básico y únicamente entiende el protocolo HTTP, de manera que ningún otro protocolo o variante de HTTP podrá funcionar

b A diferencia del proxy configurable (ProxyConf) recibe pares [IP, puerto] que solo tienen utilidad en una URL cuando el destino es un servidor web

c El supuesto es falso: el proxy entiende HTTP porque funciona a un nivel de red inferior

d En este curso no hemos estudiado cómo construir un cliente que emita órdenes en un nivel de red diferente de HTTP

31 *Supongamos el siguiente código*

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

En el código original de proxy.js, que emplea otros nombres para los sockets, el callback de la operación connect incluye unas cuantas instrucciones que tienen como propósito:

a Copiar los mensajes que llegan por s2 al socket s1 y viceversa.

b Crear un socket al que enviar las peticiones que llegan por s1, devolviendo la respuesta al cliente por s2.

c Atender al evento message en cada socket, procesando el contenido de formato JSON, y redirigiéndole de un socket a otro.

d Crear un socket al que enviar las peticiones que llegan por s2, devolviendo la respuesta al cliente por s1.

32 *En la práctica 1, consideremos la siguiente extensión al programa proxy.js original, que intenta implementar la variante del proxy programable:*

```
const net = require('net');
const LOCAL_PORT  = 8000;
const LOCAL_IP  = '127.0.0.1';
const LOCAL_PROG_PORT  = 8001;
let REMOTE_PORT = 80;
let REMOTE_IP = '158.42.4.23'; // www.upv.es
const server = net.createServer(function (socket) {
  const serviceSocket = new net.Socket();
  serviceSocket.connect(parseInt(REMOTE_PORT),
   REMOTE_IP, function () {
    socket.on('data', function (msg) {
      serviceSocket.write(msg);
    });
    serviceSocket.on('data', function (data) {
      socket.write(data);
    });
  });
}).listen(LOCAL_PORT, LOCAL_IP);
const pserver = net.createServer(function (socket)
{
  var BUFFER = '';
  socket.on('data', function (msg) {
    BUFFER = BUFFER + msg.toString();
  });
  socket.on('end', function () {
    const prog = JSON.parse(BUFFER);
    REMOTE_PORT = prog.remote_port;
    REMOTE_IP = prog.remote_ip;
  });
}).listen(LOCAL_PROG_PORT, LOCAL_IP);
```

¿Cuál de los siguientes efectos ocurre cuando este proxy recibe un mensaje del programador?

a Este proxy aborta debido a un error, ya que no permite modificar los parámetros relativos al servidor remoto.

b Se rompen las conexiones existentes.

c Las conexiones existentes se redirigen al nuevo servidor que hemos programado.

d Las conexiones posteriores se redirigen al servidor que hemos programado, pero las existentes no se ven afectadas.

TSR Primer Parcial 28 Oct 2021 **A**

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

Utiliza lápiz, y no taches una posible respuesta incorrecta: bórrala o cúbrela con typex

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

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	**25**	A	B	C	D
6	A	B	C	D	**26**	A	B	C	D
7	A	B	C	D	**27**	A	B	C	D
8	A	B	C	D	**28**	A	B	C	D
9	A	B	C	D	**29**	A	B	C	D
10	A	B	C	D	**30**	A	B	C	D
11	A	B	C	D	**31**	A	B	C	D
12	A	B	C	D	**32**	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					