TSR **Primer Parcial** **Temes 1,2,3 (Primera mitad) Lab 1** **2023-11-02** **A**

Aquest examen consta de 26 qüestions, amb una puntuació total de 10 punts. Cada qüestió té 4 alternatives, de les quals únicament una és certa. La nota es calcula de la següent manera: despres de descartar les duess pitjors qüestions, cada encert suma 10/24 punts, y cada error desconta 10/72 punts. Has de contestar en el full de respostes.

1 *A l'àrea d'aplicació de la informàtica cooperativa:*

- **a** Els nodes clients realitzen les tasques de còmput.
- **b** Els nodes servidors realitzen tasques de distribució de dades.
- **c** Totes les altres opcions són veritables.
- **d** Les fallades dels clients es poden superar fàcilment: ja sigue redistribuint les seues tasques o reenviant cada tasca a diversos clients.

2 *Quina d'aquestes oracions és certa sobre els sistemes LAMP?*

- **a** Tots els serveis SaaS han de ser sistemes LAMP.
- **b** Alguns sistemes LAMP utilitzen Windows 11 com a sistema operatiu.
- **c** Els sistemes LAMP utilitzen, entre altres elements, servidors web Apache.
- **d** Els sistemes LAMP estan dissenyats per a una alta escalabilitat.

3 *Per millorar l'escalabilitat del seu servei de base de dades intern, Wikipedia:*

- **a** Utilitza un sistema de gestió de bases de dades NoSQL implementat en JavaScript.
- **b** Utilitza un model de replicació passiva en què les rèpliques secundàries poden atendre sol·licituds de només lectura.
- **c** No utilitza replicació, per evitar condicions de carrera i inconsistències.
- **d** Utilitza una política d'administració que impedeix als usuaris actualitzar el contingut de la base de dades.

4 *La programació asincrònica (PA) presenta aquest avantatge en comparació amb la programació concurrent (és a dir, multifil):*

- **a** Els recursos s'utilitzen de forma seqüencial. Això evita condicions de carrera.
- **b** Quan es considera un únic procés, PA sempre executa en paral·lel totes les seves tasques.
- **c** PA no està orientada a esdeveniments: els processos PA no admeten esdeveniments externs.
- **d** Totes les altres opcions són falses.

5 *A la computació al núvol, l'objectiu principal del model de servei IaaS és:*

- **a** Automatitzar l'escalat del servei.
- **b** Automatitzar l'actualització del servei.
- **c** Proporcionar serveis de programari específics als seus clients.
- **d** Proporcionar una infraestructura adequada per desplegar aplicacions distribuïdes.

6 *Volem mostrar el contingut del fitxer Example.txt (ubicat al directori actual i amb permís de lectura) a la pantalla usant promeses. Una possible implementació a Node.js és:*

- **a** Cap, ja que aquesta funcionalitat no es pot implementar mitjançant promeses.
- **b**

```
const fs=require('fs')
console.log(fs.readFileSync("Example.txt",'utf8'))
```

- **c**

```
const fs=require('fs').promises
fs.readFile("Example.txt",'utf8').then(console.log)
```

- **d**

```
const fs=require('fs').promises
fs.readFile("Example.txt",'utf8').catch(console.log)
```

Consideremos estos programas JavaScript:

```
// Program: ex1.js
function f(x) {
   return (y) => { x++; return x+y }
}
```

```
// Program: ex2.js
function f(x) {
   return (y) => { x++; return x+y }
}
g=f(0)
console.log(g(2))
console.log(g(3))
```

```
// Program: ex3.js
function f(x) {
   return (y) => { x++; return x+y }
}
g=f(0)
h=f(0)
console.log(g(2))
console.log(h(2))
```

```
// Program: ex4.js
const ev = require('events')
const emitter = new ev.EventEmitter()
const e1 = "print"
emitter.on(e1, (y) => {console.log( y+"!!") })
setTimeout(() => {emitter.emit(e1, "First")}, 2000)
emitter.emit(e1,"Second")
console.log("End.")
```

7 *Quin és el resultat d'una execució de ex4.js?*

a First!!
Second!!
End.

b End.
First!!
Second!!

c Second!!
End.
First!!

d End.
Second!!
First!!

8 *Què es veu a la pantalla al executar ex2.js?*

a Un error.

b 1y
2y

c 2
3

d 3
5

9 *Afegim a ex1.js una crida f(5) després de la seva última línia. Quin és el resultat d'aquesta crida?*

a Una funció.

b Valor 6.

c Aquesta cadena: 5+y.

d Un error.

10 *Quin és l'àmbit de x al programa ex2.js?*

a Global

b Local per a f i inaccessible per a la funció retornada per f.

c Local per a f i accessible a la clausura de g.

d Cap: el seu primer accés avorta el procés.

11 *Quantes vegades es passa una funció com a argument a ex4.js?*

a Una.

b Dos.

c Tres.

d Cap.

12 *Què es veu a la pantalla al executar ex3.js?*

a Un error

b 1y
1y

c 3
4

d 3
3

13 *Considerem aquest programa JavaScript:*

```
"use strict"
for (var i=0; i<5; i++) {
  console.log ("i: " + i)
}
console.log ("end --> i=" + i)
```

Si eliminem la paraula clau var a la segona línia, quins són els canvis, si n'hi ha, en l'execució del programa resultant?

a El programa s'executa de la mateixa manera.

b El programa avorta a la seva segona línia.

c El programa avorta a la seva última línia.

d Cap de les altres opcions és correcta.

14 *Què es mostra quan s'executa aquest programa?*

```
function f1 (a,b,c) {
  console.log (arguments.length + "arguments")
  return a+b+c;
}
console.log( "result: " + f1 (3))
```

a 3 arguments
result: 3undefinedundefined

b 3 arguments
result: 3

c 1 arguments
result: 3

d 1 arguments
result: 3undefinedundefined

15 *El terme comunicació no persistent, a l'àrea dels sistemes de missatgeria, significa:*

a Comunicació no fiable; és a dir, es poden perdre missatges.

b L'emissor i el receptor no es bloquegen en la gestió de comunicacions.

c Els canals de comunicació no tenen capacitat i això obliga els dos agents, emissor i receptor, a estar disponibles i connectats abans d'iniciar el seu intercanvi de missatges.

d Els agents receptors bloquegen la seva operació de recepció quan no hi ha missatges al canal de comunicació.

16 *Considerem aquest programa JavaScript:*

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

console.log("Call to first readFile")
fs.readFile("/proc/loadavg",(e,d)=> {
  if (e) console.error(e.message)
  else console.log(d+")
  console.log("End of first readFile\n")
})
console.log("Call to second readFile")
console.log( fs.readFileSync("/proc/loadavg") + "" )
console.log("End of second readFile\n")
```

Quina és l'última línia de text que mostra aquest programa?

a L'última línia a /proc/loadavg

b End of first readFile

c End of second readFile

d Pot canviar d'una execució a una altra.

17 *Aquesta és una variació breu del programa emitter2.js utilitzat a la pràctica 1:*

```
const ev = require ( 'events' )
const emitter = new ev . EventEmitter ()
function handler ( event , n ) {
  return (incr)=>{
    n+=incr
    console.log(event + ': ' + n)
  }
}
emitter.on('e1', handler('e1','prefix'))
for (let i=1; i<3; i++) emitter.emit('e1',i)
```

Quin és el resultat en pantalla d'una execució d'aquest programa?

a e1: prefix1
e1: prefix12

b prefix: 1
prefix: 3

c event: prefixincr
event: prefixincrincr

d e1: prefixi
e1: prefixii

18 *ØMQ és un exemple d'aquest tipus de sistema de missatgeria:*

a Dèbilment persistent i sense broker.

b Basat en broker i fortament persistent.

c Basat en broker amb comunicació no persistent.

d Sense broker i amb comunicació no persistent.

19 *Considerem el programa proxy bàsic utilitzat a la sessió 3 del Laboratori 1:*

```
1  const net = require('net')
2  const PORT_A = 8000
3  const IP_A = '127.0.0.1'
4  const PORT_B = 80
5  const IP_B = '158.42.4.23' //www.upv.es
6  const server = net.createServer(socket=> {
7      const ss = new net.Socket()
8      ss.connect(parseInt(PORT_B),
9          IP_B, () => {
10             socket.on('data', msg => {
11                 ss.write(msg)
12             })
13             ss.on('data', data => {
14                 socket.write(data)
15             })
16     })
17 })
18 server.listen( PORT_A , IP_A )
19 console.log("accept conn on: "+PORT_A)
```

Hauríem d'estendre'l, rebent els valors de l'adreça remota i del port remot com a arguments de la línia d'ordres, per construir un proxy configurable. Quins són els canvis de programa necessaris per a aquesta fi?

a No és possible realitzar una modificació breu. Cal afegir més linies al programa

b Les línies 2 i 3 s'han de canviar a:

```
const PORT_A=parseInt(process.argv[3]+"")
const IP_A = process.argv[2]+""
```

c La línia 18 s'ha de canviar a:

```
server.listen( PORT_B, IP_B)
```

d Les línies 4 i 5 s'han de canviar a:

```
const PORT_B=parseInt(process.argv[3]+"")
const IP_B = process.argv[2]+""
```

20 *L'última tasca de la sessió 3 de la pràctica 1 consisteix a estendre el proxy configurable per construir un proxy programable. Aquest proxy programable rep el port i l'adreça IP inicials del servidor remot des de la línia d'ordres, però també pot acceptar valors nous d'un procés programador. Per construir un proxy tan evolucionat, necessitem aplicar aquests canvis (entre d'altres) al programa del proxy configurable:*

a Accedir i processar de manera adequada dos arguments addicionals des de la línia d'ordres.

b Crear un socket client TCP addicional, connectar-lo al servidor remot i enviar a través d'aquest nou socket tota la informació rebuda dels processos clients.

c Crear un altre socket servidor que escolte les connexions del procés programador, actualitzant els valors REMOTE_PORT i REMOTE_IP amb la informació rebuda.

d Crear un socket addicional amb net.createServer() que escolte les connexions del procés programador i reenvie tots els missatges que rep al servidor remot.

21 *A la documentació de ØMQ, el terme missatge segmentat (o missatge multipart) es refereix a un tipus concret d'estructura de missatge que exigeix una gestió d'enviament i recepció diferent a la dels missatges no segmentats. Suposem que 'so' és un socket. Seleccioneu l'opció que proporciona un exemple d'enviament d'un missatge segmentat en ØMQ:*

a

```
so.send(["seg1","seg2"])
```

b

```
so.send("seg1","seg2","seg3")
```

c

```
so.send(JSON.stringify({seg1:valor1,seg2:valor2}))
```

d

```
so.on("message", (s1,s2)=>console.log(s1+s2))
```

22 *Suposem que s'utilitzarà una URL A (per exemple, A = tcp://158.42.1.118:30000) per interconnectar múltiples processos en ØMQ. La següent oració és veritable:*

a Totes les crides a connect(A) han de precedir a la primera crida a bind(A).

b La crida a bind(A) ha de precedir totes les crides a connect(A).

c No hi ha problema si un procés crida a connect(A) i després un altre procés fa la primera crida a bind(A).

d Diferents processos poden fer crides simultànies i correctes a bindSync(A).

23 *Considerem el següent parell de programes Node.js que usen ØMQ:*

```
// Program: publisher.js
const zmq = require("zeromq")
const pub = zmq.socket('pub')
let count = 0
pub.bindSync("tcp://*:5555")
setInterval(function() {
  pub.send("TEST " + count++)
}, 1000)
```

```
// Program: subscriber.js
const zmq = require("zeromq")
const sub = zmq.socket('sub')
sub.connect("tcp://localhost:5555")
sub.subscribe("TEST")
sub.on("message", function(msg) {
  console.log("Received: " + msg)
})
```

Si totes les instàncies d'aquests programes s'executen al mateix ordinador …

a Només podem iniciar un únic emissor a cada execució d'aquest conjunt de programes.

b Totes les altres opcions són veritables.

c Si eliminem al programa subscriptor la crida a sub.subscribe, no hi haurà cap canvi en el comportament dels processos resultants.

d Podem iniciar, en qualsevol ordre, diversos subscriptors i un sol emissor. El conjunt de processos resultant no generarà cap error i tots els missatges enviats s'entregaran a tots els subscriptors eventualment.

24 *Considerem el següent parell de programes Node.js que usen ØMQ:*

```
// Program: sender.js
const zmq = require("zeromq")
const producer = zmq.socket("push")
let count = 0
producer.bind("tcp://*:8888", (err) => {
   if (err) throw err
   setInterval( () => {
        producer.send("msg# " + count++)
    }, 1000)
})
```

```
// Program: receiver.js
const zmq = require("zeromq")
const consumer = zmq.socket("pull")
consumer.connect("tcp://127.0.0.1:8888")
consumer.on("message", function(msg) {
   console.log("received: " + msg)
})
```

Si totes les instàncies d'aquests programes s'executen al mateix ordinador …

a Al programa receiver.js podem afegir, com a última línia del seu listener per a message, una instrucció consumer.send(msg). Contestarà a l'emisor.

b Totes les altres opcions són veritables.

c Si s'inicia un únic receptor en cada execució d'aquest parell de programes, l'argument per a la vostra crida a consumer.connect() pot ser tcp://*:8888.

d Podem iniciar, en qualsevol ordre, diversos receptors i un sol emissor. El conjunt de processos resultant no generarà cap error i els missatges eventualment es lliuraran.

25 *Entre altres coses, el patró de comunicació ØMQ REQ-REP es considera sincrònic perquè*

a La recepció de missatges de sol·licitud i resposta no es pot gestionar utilitzant listeners en els processos servidor i client, respectivament.

b Un socket REQ no pot enviar ni transmetre dos missatges de sol·licitud consecutius si no rep un missatge de resposta entre aquests enviaments.

c Aquest patró no pot proporcionar persistència de comunicació.

d Un servidor no pot gestionar connexions simultànies amb més d'un client.

26 *Considerem el següent parell de programes Node.js que usen ØMQ:*

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

```
// Program: server.js
const zmq = require('zeromq')
const rp = zmq.socket('rep')
rp.bind('tcp://127.0.0.1:8888',
    function(err) {
      if (err) throw err
    })
rp.on('message', function(msg) {
    console.log('Request: ' + msg)
    rp.send('World')
})
```

Si totes les instàncies d'aquests programes s'executen al mateix ordinador …

a No hi haurà cap error si iniciem i executem simultàniament dos processos servidors.

b Si eventualment iniciem un procés servidor, serà possible iniciar múltiples processos clients. Aquests clients eventualment rebran una resposta.

c Un servidor pot rebre i entregar un missatge enviat per un client abans d'enviar la resposta a una sol·licitud enviada i entregada prèviament des d'un altre client.

d Totes les altres opcions són veritables.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A

DNI NIE PASSAPORT

0 0 0 0 0 0 0 0 0

1 1 1 1 1 1 1 1 1

2 2 2 2 2 2 2 2 2

3 3 3 3 3 3 3 3 3

4 4 4 4 4 4 4 4 4

5 5 5 5 5 5 5 5 5

6 6 6 6 6 6 6 6 6

7 7 7 7 7 7 7 7 7

8 8 8 8 8 8 8 8 8

9 9 9 9 9 9 9 9 9

ETSINF - TSR

Primer Parcial - 02/11/2023

Cognoms .....

Nom .....

Marque així Així NO marque

NO ESBORRAR, corregir amb corrector

Primer Parcial

	a	b	c	d
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				