TSR **Primer Parcial** **Temes 1,2,3 (1a part) Pràctica 1** **2022-11-03** **A**

Aquest examen té un valor de 10 punts, i consta de 24 qüestions amb 4 alternatives i una única resposta correcta. Cada resposta correcta aporta 10/24 punts, cada error descompta 10/72 punts. Conteste en la fulla de respostes.

1 *En la computació cooperativa:*

- **a** Les tasques de còmput solen realitzar-se en els *clients,* mentre que el *servidor* reparteix les tasques a realitzar.
- **b** La fallada d'un client impedirà que la computació global acabe.
- **c** Els clients solen intercanviar resultats parcials entre ells.
- **d** Totes les altres opcions són certes.

2 *En analitzar el cicle de vida dels serveis, el rol* administrador de sistema *se centra en:*

- **a** Desenvolupar les aplicacions sol·licitades pels usuaris.
- **b** Decidir quins components formaran part d'una aplicació distribuïda que proporcione el servei sol·licitat.
- **c** Decidir en quins nodes s'instal·larà cada component i amb quina configuració, així com comprovar que cada ordinador funcione correctament.
- **d** Cap de les altres opcions és certa.

3 *Un avantatge del model de programació asincrònica quan es compara amb la programació concurrent basada en múltiples fils d'execució és:*

- **a** Es redueix en gran manera la probabilitat que hi haja condicions de carrera.
- **b** Millora el rendiment: no sols llança diferents fils, sinó que aquests fils no passen mai a estat suspès.
- **c** Per utilitzar comunicació no persistent, els clients no necessiten connectar-se a cap servidor abans d'enviar les seues peticions.
- **d** Totes les altres opcions són vàlides.

4 *En la Wikipedia, la replicació de components …:*

- **a** No resulta necessària en cap dels components del sistema.
- **b** Sol utilitzar-se en els seus components, combinada amb altres mecanismes com puga ser l'ús de memòries cau i la distribució o sharding de dades.
- **c** S'utilitza únicament en el component *Apache.*
- **d** S'utilitza únicament en el component *MySQL.*

5 *Considere el següent 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")
```

Quina serà l'última cadena que aquest programa mostrarà, si el fitxer que s'intenta llegir existeix, té contingut, i no hi ha errors en les seues lectures?

- **a** `End of asynchronous readFile`
- **b** El contingut del fitxer `/proc/loadavg`
- **c** `End of synchronous readFile`
- **d** Cap de les altres opcions és correcta.

Considere's el següent conjunt de programes 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 (i) {
    let z = x + i
    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 *Què mostrarà en pantalla l'execució de* `ex1.js`*?*

- **a** Mostrarà Event print: 1
- **b** Mostrarà una successió de missatges:
  Event print: 1
  Event print: 2
  Event print: 3
  …
- **c** Es generarà una excepció en executar la línia `emitter.emit(e1)`.
- **d** No arribarà a mostrar-se cap missatge.

7 *Quantes funcions anònimes (no confondre amb crides a funcions) es defineixen en* `ex1.js`*?*

- **a** Cap de les altres opcions és certa.
- **b** Infinites, perquè es dona una definició recursiva.
- **c** Una.
- **d** Dos.

8 *Quin valor retorna una crida a la funció* `f` *amb l'argument* `10`*, és a dir,* `f(10)`*, en el programa* `ex2.js`*?*

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

9 *Podria el programa* `server.js` *mantenir múltiples connexions de processos clients?*

- **a** No podria, perquè JavaScript només té un fil d'execució i cada connexió amb un client necessita ser gestionada per un fil diferent.
- **b** Podria, perquè tant l'establiment i el tancament de la connexió, com l'arribada de nous missatges o l'ocurrència d'errors en la connexió es modelen com a esdeveniments i JavaScript disposa d'una cua d'esdeveniments per a gestionar-los, hi haja els que hi haja.
- **c** No podria, perquè només existeix un callback per a gestionar l'establiment d'una connexió i mentre aquest s'execute, no pot haver-hi una altra activitat en execució.
- **d** En principi podria, però aquest programa no arriba a fer-ho perquè no disposa de cap `listener` per a l'esdeveniment `connect`.

10 *Considere que es disposa d'una funció* `countLines(text)` *que retorna el nombre de línies contingudes en la cadena de text facilitada en el seu paràmetre* `text`. *Si es necessitara escriure un programa Node.js que utilitzara promeses i mostrara el nombre de línies contingudes en el fitxer* `Example.txt` *del directori actual (existent i amb permís de lectura), una possible solució seria:*

a Cap, perquè amb promeses no és possible realitzar aqueixa gestió.

b

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

c

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

d

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

11 *Indique l'afirmació correcta respecte a Node.js:*

a Node està implementat internament mitjançant un únic fil d'execució. És a dir Node en sí mateix és mono-fil.

b Node integra el motor de JavaScript V8, el mateix motor de JavaScript que utilitza Google Chrome.

c Node resulta adequat per a executar aplicacions escrites en JavaScript i en Java.

d Node i tota la funcionalitat dels seus mòduls està integrada en els navegadors web moderns, com Google Chrome o Firefox.

12 *Si executem el següent fragment de codi, indique l'afirmació correcta:*

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

a S'imprimirà la cadena `hello` per la consola i després s'imprimirà `bye`

b Produirà un error, perquè s'emet l'esdeveniment `hello` i no hi ha cap manejador per a aquest esdeveniment.

c S'imprimirà la cadena `bye` per la consola i després s'imprimirà `hello` .

d Si executem el programa diverses vegades, unes vegades s'imprimirà `hello` abans que `bye` i altres vegades s'imprimirà `bye` abans que `hello` .

13 *Respecte al broker (o gestor) en els sistemes de missatgeria, selecciona l'afirmació vertadera:*

a Els sistemes sense broker no són adequats quan es desitja persistència feble

b Un sistema amb broker pot ser construït a partir d'un altre sense broker

c Els sistemes amb broker faciliten, encara que no proporcionen directament, una visió d'estat compartit

d Un sistema sense broker pot ser construït a partir d'un altre amb broker

14 *ØMQ, com a sistema de missatgeria, intenta cobrir diversos objectius. Indica quin NO ho és:*

a Facilitat en el seu ús per als programadors per reproduir una API similar als sockets BSD

b Fiabilitat garantida mitjançant el seguiment automàtic dels missatges pendents de recepció

c Reutilització del mateix codi, excepte la URL, per a comunicar fils, processos en el mateix ordinador, i processos en ordinadors diferents

d Facilitat per a resoldre problemes comuns en suportar patrons bàsics d'interacció

15 *Siga un emissor el codi del qual connecta sense errors amb receptors que han efectuat l'operació bind corresponent amb 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 continuació l'emissor intenta executar aquestes 3 instruccions:

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

Ocorrerà això:

- **a** Cada receptor rep els 3 missatges en el mateix ordre en què s'envien
- **b** L'emissor no pot iniciar la invocació del segon q.send fins a rebre resposta del primer
- **c** Cada receptor rep només un dels missatges, però el socket emissor ha d'esperar resposta per a cadascun abans de propagar el següent
- **d** Cada receptor rep només un dels missatges, i el socket emissor només ha d'esperar resposta després de propagar l'últim dels tres

16 *En quines condicions múltiples sockets ØMQ de tipus REQ es poden interconnectar?*

- **a** Només quan hi haja exactament un realitzant l'operació bind
- **b** Només quan no hi haja més d'un enviament simultani pel socket
- **c** Només quan s'empre transport local (IPC o fils)
- **d** No es poden interconnectar perquè REQ necessita que en l'altre extrem s'utilitze un socket complementari

17 *S'ha desenvolupat un programa servidor que utilitza un socket REP per a interactuar amb els clients, que utilitzen sockets REQ. No obstant això, aquesta solució només permet processar una sol·licitud a cada moment i convindria millorar el seu rendiment. Per a això, un programador proposa que tant els clients com el servidor utilitzen un únic socket PUSH per a enviar missatges i un únic socket PULL per a rebre'ls. Si habitualment hi haurà múltiples clients connectats i interactuant amb el servidor, serà aquesta una solució correcta?*

- **a** No, perquè el socket PUSH no pot ser utilitzat per a enviar missatges: només permet rebre'ls.
- **b** No, perquè el socket PUSH envia els missatges seguint l'ordre rotatori de connexió i podria enviar respostes a clients que no han enviat cap petició.
- **c** Sí, perquè el patró PUSH/PULL no bloqueja en cap cas l'enviament o recepció de missatges i el rendiment millorarà clarament.
- **d** Cap de les altres afirmacions és certa.

18 *En 0MQ, en l'establiment d'una connexió:*

- **a** Cada socket només pot fer, com a molt, una operació bind.
- **b** Cada socket només pot fer, com a molt, una operació connect.
- **c** L'operació bind sempre ha de precedir a les operacions connect.
- **d** Les altres respostes són errònies.

19 *Entre les fonts de complexitat d'un sistema distribuït es troben:*

- **a** La sol·licitud de serveis.
- **b** El format de la informació.
- **c** La gestió de fallades.
- **d** Totes les altres respostes són vàlides.

20 *Indique el resultat que es veurà per la consola en executar el següent fragment de codi:*

```
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 *Considera el següent fragment de codi i indica, en els primers 4 segons (inclusivament), quantes vegades es mostrarà el text* `Esdeveniment un tractat` *per pantalla.*

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

var handler = function () {
   console.log("Esdeveniment "+e1+" tractat")
}

emitter.on(e1, handler)

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

etapa()
```

a Cap perquè hi ha un error en la funció proporcionada com a argument d'emitter.on

b Cap perquè hi ha un error en la funció proporcionada com a argument de setInterval

c 4 o 5

d Almenys 7

22 *Donat el següent programa, indique l'afirmació correcta:*

```
let MAX=1000

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

// Fer treball
for (let i=0; i<MAX; i++) {}

setTimeout ( () => console.log ("tres"), 99)
setTimeout ( () => console.log ("quatre"), 0)
console.log ("cinc")
```

a Quan ho executem, sempre veurem `cinc` abans que `quatre` .

b Quan ho executem, és possible que vegem `quatre` abans que `cinc` .

c Quan ho executem, sempre veurem `dos` abans que `tres`

d El programa imprimeix `un` i `cinc` i acaba sense imprimir res més.

23 *Considerem el següent fragment del codi del proxy invers vist en la tercera sessió de la pràctica 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)
```

*Les instruccions que falten en les línies *** són:*

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 Cap de les altres opcions és vàlida.

d Les dues opcions que mostren codi són vàlides.

24 *L'exercici del proxy programable inclou un nou servei del proxy que, resumidament, permet modificar REMOTE_PORT i REMOTE_IP. Imagina que per a fer això es proposa incorporar el següent codi al proxy programable:*

```
// s1 connecta al proxy amb el client.
// s2 connecta al proxy amb el servidor remot.
const prog = net.createServer((s3)=>{
  s3.on('data', function(m3) {
    ***
  })
})
```

I que la petició enviada per programador.js es construeix a partir d'argv de la següent forma:

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

*Les instruccions que falten en les línies *** del proxy programable proposat són:*

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 Novembre 2022 **A**

Emplena i entrega aquest full de respostes.Cada qüestió té una única resposta correcta.No oblides emplenar correctament les teues dades personals.

No has de fer cap marca damunt d´una possible resposta incorrecta: esborra-la o cobreix-la amb Typex

Una qüestió amb més d´una resposta marcada es 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: _____

Cognoms: _____

Nom: _____

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						