TSR - Primer Parcial. 2024-11-04

Aquest examen consta de 22 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: després de descartar les dues pitjors qüestions, cada encert suma 0.5 punts, i cada error descompta 1/6 punts. Has de contestar en el full de respostes.

A

ALUMN@

1. ***Pel que fa als avantatges que aporta un sistema distribuït, selecciona l'afirmació correcta***
   - **A.** Millora el rendiment
   - **B.** Permet millorar la disponibilitat
   - **C.** Facilita la compartició de recursos
   - **D.** Totes les altres afirmacions són certes

2. ***En relació amb el concepte de clúster altament disponible, aquests clústers:***
   - **A.** Sacrifiquen la disponibilitat dels servidors per mantenir la integritat de la informació gestionada
   - **B.** Requereixen programes, equips i personal especialitzat dedicats, i per tant resulta car per a les empreses
   - **C.** Cap de les altres afirmacions és certa
   - **D.** Sacrifiquen la integritat de la informació gestionada per mantenir la disponibilitat dels servidors

3. ***En relació amb la computació al núvol (Cloud Computing)***
   - **A.** Idealment, presenta la següent estructura en nivells: SaaS, PaaS, IaaS
   - **B.** Totes les altres afirmacions són certes
   - **C.** És possible gràcies a l'ús de la tecnologia de virtualització d'equips
   - **D.** Introdueix un model de "pagament per ús"

4. ***Suposem definit el següent programa, anomenat copy.js***

```
const fs=require('fs')
function copyFile(source, dest) {
  console.log("reading")
  fs.readFile(source, (error, data)=> {
    if (!error) {
    console.log("writing")
    fs.writeFile(dest, data, error=>{
      if (!error) console.log("done")
    })
    }
  })
}
copyFile("copy.js","copy2.js")
console.log("start")
```

Suposant que la lectura i escriptura de fitxers no generen errors (hi ha el fitxer a llegir, tenim els permisos necessaris, hi ha espai per escriure el nou fitxer, etc.), indica el resultat esperat en executar el programa

- **A.** Error sintàctic en intentar executar el programa
- **B.** Escriu en pantalla (en aquest ordre) les línies

```
reading
start
writing
done
```

- **C.** Escriu en pantalla (en aquest ordre) les línies

```
start
reading
writing
done
```

- **D.** Escriu en pantalla les línies (`start`, `reading`, `writing`, `done`), però no podem saber l'ordre (pot ser un ordre diferent a cada execució)

5. ***Suposem definit el següent programa, anomenat pcopy.js***

```
const fs=require('fs').promises
function copyFile(source, dest) {
  console.log("reading")
  fs.readFile(source)
  .then(data=>{console.log("writing");
         return fs.writeFile(dest, data)})
  .then(()=>console.log("done"))
}
copyFile("pcopy.js","pcopy2.js")
console.log("start")
```

Suposant que la lectura i escriptura de fitxers no generen errors (hi ha el fitxer a llegir, tenim els permisos necessaris, hi ha espai per escriure el nou fitxer, etc.), indica el resultat esperat en executar el programa

A. Escriu en pantalla (en aquest ordre) les línies

```
reading
start
writing
done
```

B. Escriu en pantalla (en aquest ordre) les línies

```
start
reading
writing
done
```

C. Escriu en pantalla les línies (`start`, `reading`, `writing`, `done`), però no podem saber l'ordre (pot ser un ordre diferent a cada execució)

D. Error sintàctic en intentar executar el programa

6. ***A continuació es presenta un programa client i un programa servidor:***

```
***cliente
let net=require('net');
let sock=net.connect({port:3000}, ()=>{
  for (let i=1; i<=10; i++) sock.write('hello');
});
sock.on('data', (x)=>{console.log(""+x)});

***servidor
const net=require('net');
const server=net.createServer( (sock)=>{
   sock.on('data', (x)=>sock.write('OK '+x));
}).listen (3000);
```

En relació amb aquests dos programes, seleccioneu l'opció correcta:

A. El servidor acaba en respondre tots els missatges que envia el client

B. Ni el programa client ni el programa servidor acaben

C. Cap de les altres afirmacions és certa

D. El client acaba en rebre tots els missatges que envia el servidor

7. ***Considereu el següent programa JavaScript:***

```
function f(x) {
  return function (y) {
    x++
    return x+y
  }
}
g=f(5, 10)
console.log(g(3))
```

Indiqueu què mostraria en pantalla aquest programa durant la seva execució.

A. 14

B. 63

C. 9

D. Un error per haver invocat f() amb més d'un argument

8. ***Considereu el següent programa JavaScript:***

```
setTimeout(()=>{console.log("1")},10)
let k=Math.abs(parseInt(process.argv[2]))
let j=0
for (let i=0; i<k; i++) j+=i
setTimeout(()=>{console.log("2")}, 1)
```

Indiqueu què mostraria en pantalla aquest programa durant la seva execució si se li passa un valor numèric enter com a primer argument a la línia d'ordres i cap de les operacions aritmètiques genera un error.

A. Els valors 1 i 2, cadascun en una línia, i el seu ordre dependrà del valor facilitat a l'argument

B. Sempre una primera línia amb un 1 i una segona línia amb un 2

C. Sempre una primera línia amb un 2 i una segona línia amb un 1

D. Un error, ja que no es pot utilitzar més d'una vegada `setTimeout()` en un mateix programa

9. ***Considereu el següent programa JavaScript:***

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

Aquest programa mostra com es podria escriure un servidor molt senzill que utilitze un socket TCP. Com aconsegueix el procés que execute aquest programa atendre múltiples connexions dels clients si només disposa d'un únic fil d'execució?

A. Permetent que cada connexió rebuda siga gestionada per una crida a la funció definida com a únic argument de `createServer()`

B. Gestionant les arribades de missatges en aquestes connexions mitjançant esdeveniments `"data"`

C. Utilitzant la cua d'esdeveniments per seqüenciar l'execució de tots els listeners que es vagen activant durant l'ús d'aquestes connexions

D. Totes les altres afirmacions són certes

10. ***Considereu el següent programa JavaScript:***

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

Aquest programa mostra com es podria escriure un servidor molt senzill que utilitze un socket TCP. Quin missatge mostrarà en primer lloc (en cas que no hi haja cap error) un procés que execute aquest programa?

A. `"server connected"`

B. `"server bound"`

C. O bé `"server bound"` o `"server connected"`

D. Qualsevol dels següents: `"data from client..."`, `"server connected"` o `"server bound"`

11. ***Considereu el següent programa client que utilitza ZeroMQ per connectar-se a dos servidors:***

```
const zmq=require('zeromq')
const rq=zmq.socket('req')
rq.connect('tcp://127.0.0.1:8888')
rq.connect('tcp://127.0.0.1:8889')
rq.send('Hello!')
rq.send('Hello again!')
rq.on('message', function (msg) {
  console.log('Response: '+msg)
})
```

Seleccioneu l'afirmació correcta sobre el funcionament del programa:

- **A.** Si no hi ha cap servidor escoltant al port 8889, aquest client no arriba a mostrar res en pantalla, ja que la seva execució es bloquejarà abans del `rq.on()`
- **B.** El codi d'aquest client envia la cadena `"Hello!"` al servidor que gestione el port 8888 local i la cadena `"Hello again!"` al servidor que gestione el port 8889 local
- **C.** Aquest programa no funcionarà correctament, ja que necessitaria un segon `rq.on('message'...)` per processar la resposta del segon servidor
- **D.** L'execució del procés que execute aquest programa finalitza tan prompte com haja rebut la primera resposta per part d'algun servidor

12. ***Els components d'un sistema distribuït:***

- **A.** Totes les altres afirmacions són certes
- **B.** Han de poder ser especificats i desenvolupats de manera independent
- **C.** Han de poder ser desplegats autònomament
- **D.** Han de poder interactuar de manera senzilla i útil

13. ***Els estàndards faciliten la superació de les complexitats, però aquestes NO inclouen***

- **A.** L'especificació de la funcionalitat dels serveis
- **B.** El format de la informació transmesa
- **C.** La sincronització entre client i servidor
- **D.** La diferenciació de rols exercits en un sistema informàtic

14. ***Pel que fa als sistemes de missatgeria:***

- **A.** A les versions persistents el receptor disposa d'un buffer per a missatges, que es poden enviar fins i tot després de finalitzar l'emissor
- **B.** Les versions no persistents amb broker no tenen sobrecàrrega per l'ús del broker
- **C.** A les versions no persistents sense broker l'emissor i receptor han de mantindre els missatges en memòria
- **D.** A les versions no persistents s'exigeix que només es puga transmetre el missatge si el receptor està actiu

15. ***Usant ZeroMQ s'ha desenvolupat el codi de dos components, que permet la comunicació en dos ordinadors d'una xarxa IP. Es faran canvis, i es pot reutilitzar la majoria del codi anterior... :***

- **A.** Si el canvi no suposa que aquests components passen a executar-se al mateix equip
- **B.** Cap de les altres afirmacions és correcta
- **C.** Només si el llenguatge de programació de tots dos components és el mateix
- **D.** Si el canvi no suposa que aquests components esdevinguen fils d'un mateix procés

16. ***Un dels avantatges de l'ús de ZeroMQ per programar components que es comuniquen és:***

- **A.** Pots triar diferents llenguatges per programar aquests components
- **B.** Ja no cal sockets ni la semàntica de les operacions que els acompanyen
- **C.** Pots comunicar components ZeroMQ amb altres que facen servir el mòdul `net` de NodeJS
- **D.** Pots comunicar components ZeroMQ amb altres que facen servir el mòdul `http` de NodeJS

17. ***Suposem que en un equip X disposem de tres components A, B i C que fan servir ZeroMQ. Indica quina afirmació és certa:***

- **A.** Si A i B són processos que es comuniquen entre si, calen dues biblioteques de ZeroMQ a l'equip X
- **B.** Si A i B es comuniquen entre si, i C ho fa amb un component d'un altre equip Y, calen dues biblioteques de ZeroMQ a l'equip X
- **C.** Si A, B i C són fils d'un mateix procés que es comuniquen entre si, calen tres biblioteques de ZeroMQ a l'equip X
- **D.** Només cal una biblioteca de ZeroMQ a l'equip X, independentment del nombre de components que es comuniquen

18. ***Considereu el següent programa JavaScript:***

```
for (let i=0; i<10; i++) {
  var j=i;
  setTimeout(()=>console.log("j = "+j), j*1000);
}
// Final del programa
```

En aquest codi es programen 10 esdeveniments, utilitzant les variables i i j. Sobre aquestes variables podem dir que:

A. A l'inici del programa, no hi ha cap variable declarada

B. Al final del programa, j té el valor 0, i s'imprimeix la seqüència:`"j = 0"`, `"j = 1"`, `"j = 2"... "j = 9"` (en línies separades)

C. Cap de les altres afirmacions és certa

D. Al final del programa, cap variable està declarada (present a l'àmbit), i s'imprimeix `"j = 9"` diverses vegades

19. ***Donat el codi següent, trieu l'afirmació correcta:***

```
let fg
{
  let x=1
  fg= (y)=>{console.log("hello "+y+" "+ x++)}
}
for (let i=0; i<10; i++) fg("pp")
```

A. El codi és incorrecte ja que la funció `fg` no es pot utilitzar fora del bloc

B. En executar el programa s'imprimiran per la consola 10 línies idèntiques

C. La variable `x` forma part d'una clausura

D. No hi ha clausures, ja que no hi ha funcions que retornen funcions

20. ***Considera el següent programa, similar als programes emitterX.js usats al laboratori 1:***

```
const ev=require('events')
const emitter=new ev.EventEmitter()
const e1='e1', e2='e2', e3='e3'
let inc=1

function handler(e,n) {
  return (inc)=> {
    n+=inc
    console.log(e + ' --> ' + n)
  }
}

emitter.on(e1, handler(e1,1))
emitter.on(e2, handler(e2,'2'))
emitter.on(e3, handler(e3,3))
emitter.on(e3, handler(e3,'3'))

function phase() {
  emitter.emit(e1,inc)
  emitter.emit(e2,0)
  inc++
}

setInterval(phase,1000)
setTimeout(process.exit,2500)
```

Quina és l'eixida en pantalla en executar el programa?

A.
```
e1 --> 2
e2 --> 20
e3 --> 4
e3 --> 31
e1 --> 4
e2 --> 200
e3 --> 6
e3 --> 312
```

B.
```
e1 --> 2
e2 --> 20
e1 --> 4
e2 --> 200
```

C.
```
e1 --> 2
e2 --> 21
e1 --> 4
e2 --> 212
```

D.
```
e1 --> 2
e2 --> 21
e3 --> 4
e3 --> 31
e1 --> 4
e2 --> 212
e3 --> 6
e3 --> 312
```

21. ***A la part final de la pràctica 1 es formula una qüestió en què, atès un pool de servidors, el proxy ha d'enviar la petició al servidor menys carregat del pool.***
   A. Es pot aconseguir modificant únicament el codi de programador.js
   B. Es pot aconseguir sense afegir ni modificar codi als equips servidors
   C. Es pot aconseguir modificant únicament el codi de `ProxyProg.js` (o `ProxyConf.js`)
   D. Caldrà afegir (o modificar) codi als equips servidors

22. ***Per implementar el proxy programable de la pràctica 1...***
   A. Ja no cal crear un socket específic per dialogar amb el servidor (`serverSocket`)
   B. Utilitzem `JSON.stringify` i `JSON.parse` per codificar/decodificar els missatges entre el client i el proxy
   C. Al proxy hem d'invocar dues vegades `net.createServer`: una per definir el codi que atén el client, i una altra per definir el codi que atén el servidor
   D. Al proxy hem d'invocar dues vegades `net.createServer`: una per definir el codi que atén el client, i una altra per definir el codi que atén el programador