This exam is worth 10 points, and consists of 26 questions. Each question poses 4 alternatives and has only one correct answer. Once discarded the two worst answers, each correct answer earns 10/24 points, and each error deducts 10/72 points. You must answer on the answer sheet.

1 *In the cooperative computing application area:*

a Client nodes carry out the computing tasks.

b Server nodes carry out data distribution tasks.

c All other choices are true.

d Client failures may be easily overcome: either redistributing their tasks or forwarding every task to multiple clients.

2 *Which of these sentences about LAMP systems is true?*

a All SaaS services must be LAMP systems.

b A few LAMP systems use Windows 11 as their operating system.

c LAMP systems use, among other elements, Apache web servers.

d LAMP systems are designed for highly scalability.

3 *In order to enhance the scalability of its internal database service, Wikipedia uses:*

a A NoSQL database management system implemented in JavaScript.

b A passive replication model in which back-up replicas may serve read-only requests.

c No replication, in order to avoid race conditions and inconsistencies.

d A management policy that prevents users from updating the database contents.

4 *Asynchronous programming (AP) introduces this advantage when it is compared with concurrent (i.e. multi-threaded) programming:*

a Programs run in a sequential way. This avoids race conditions.

b Race conditions often arise.

c AP is not event-oriented; thus, AP processes cannot be interrupted by external events.

d All other choices are false.

5 *In cloud computing, the main goal of the IaaS service model is:*

a Automate service scaling.

b Automate service upgrading.

c Provide specific software services to its customers.

d Provide an adequate infrastructure for deploying distributed applications.

6 *Let us show the contents of file Example.txt (located in the current directory and with read permission for our user) on the screen using promises. A possible implementation in Node.js is:*

a None, since that functionality cannot be implemented using promises.

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)
```

Let us consider these JavaScript programs:

```
// 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 *What is the output of an execution of ex4.js?*

a First!!
Second!!
End.

b End.
First!!
Second!!

c Second!!
End.
First!!

d End.
Second!!
First!!

8 *What is shown on the screen when we run ex2.js?*

a An error.

b 1y
2y

c 2
3

d 3
5

9 *Let us consider ex1.js, with a call f(5) after its last line. Which is the result of that call?*

a A function.

b Value 6.

c This string: 5+y.

d An error.

10 *What is the scope of x in program ex2.js?*

a Global

b Local to f and inaccessible to the function returned by f.

c Local to f and accessible in the closure of g.

d None, since its access aborts the process.

11 *How many times a function is passed as an argument in ex4.js?*

a One.

b Two.

c Three.

d None.

12 *What is shown on the screen when we run ex3.js?*

a An error

b 1y
1y

c 3
4

d 3
3

13 *Let us consider this JavaScript program:*

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

If we remove the var keyword in its second line, which are the changes, if any, in the execution of the resulting program?

- **a** The program runs in the same way.
- **b** The program aborts in its second line.
- **c** The program aborts in its last line.
- **d** No other choice is correct.

14 *What is displayed when this program runs?*

```
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 *The term transient communication (as opposed to persistent communication), in the area of messaging systems, means:*

- **a** Unreliable communication, i.e., messages may be lost.
- **b** Sender and receiver do not block in the communication management.
- **c** Communication channels have no capacity and this compels both agents, sender and receiver, to be ready and connected before starting their message exchange.
- **d** Receiver agents block in their receive operation when there are no messages in the communication channel.

16 *Let us consider this JavaScript program:*

```
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")
```

Which is the last text line shown by that program?

- **a** The last line in /proc/loadavg
- **b** End of first readFile
- **c** End of second readFile
- **d** It may change from one execution to the next.

17 *This is a short variation of the emitter2.js program used in Lab 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)
```

What is the screen output of an execution of that program?

a e1: prefix1
e1: prefix12

b prefix: 1
prefix: 3

c event: prefixincr
event: prefixincrincr

d e1: prefixi
e1: prefixii

18 *ØMQ is an example of this kind of messaging system:*

a Weakly persistent with no broker.

b Broker-based and strongly persistent.

c Broker-based with transient communication.

d Brokerless with transient communication.

19 *Let us consider the basic proxy program used in session 3 of Lab 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)
```

We want to extend it, receiving the remote address and remote port values as command line arguments, to build a configurable proxy. Which are the program changes needed to this end?

a No short update is possible. We need to add more lines to the program.

b Lines 2 and 3 should be changed, with these new contents:

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

c Line 18 should be changed, with this new content:

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

d Lines 4 and 5 should be changed, with these new contents:

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

20 *The last task in session 3 of Lab 1 consists in extending the configurable proxy to build a programmable proxy. That programmable proxy receives the initial port and IP address of the remote server from the command line, but it is also able to accept new values from a programmer process. In order to build such an evolved proxy, we need to apply these changes (among others) to the configurable proxy program:*

- **a** Access and process in an appropriate way two additional arguments from the command line.
- **b** Create an additional TCP client socket, connect it to the remote server and send through this new socket all the information received from client processes.
- **c** Create another server socket that listens to connections from the programmer process, updating the REMOTE_PORT and REMOTE_IP values with the incoming information.
- **d** Create an additional socket with net.createServer() that listens to connections from the programmer process, forwarding all incoming messages to the remote server.

21 *In the ØMQ documentation, the term segmented message (or multi-part message) refers to a concrete type of message structure that demands a different sending and reception handling when it is compared with non-segmented (or single-part) messages. Let us assume that `so´ is a socket. Select the option that provides an example of sending a segmented message in ØMQ:*

a

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

b

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

c

```
so.send(JSON.stringify({seg1:value1,seg2:value2}))
```

d

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

22 *Let us assume a URL A (e.g., A = tcp://158.42.1.118:30000) to be used in order to interconnect multiple processes in ØMQ. The following sentence is true:*

- **a** All calls to connect(A) must precede the first call to bind(A).
- **b** The call to bind(A) must precede all calls to connect(A).
- **c** There is no problem if a process calls connect(A) and later another process makes the first call to bind(A).
- **d** There may be multiple successful concurrent calls to bindSync(A), requested by different processes.

23 *Let us consider the following pair of Node.js programs that use 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)
})
```

All instances of those programs run in the same computer. Choose the true statement:

- **a** We may only start a single publisher in each execution of this set of programs.
- **b** All other choices are true.
- **c** If we remove in the subscriber program the sub.subscribe call, no change will arise in the behaviour of the resulting programs.
- **d** We may start, in any order, multiple subscribers and a single publisher. The resulting set of processes will not generate any error, and all sent messages will be eventually delivered to all subscribers.

24 *Let us consider the following pair of Node.js programs that use 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)
})
```

All instances of those programs run in the same computer. Choose the true statement:

- **a** In program receiver.js we may add, as the last line of its message listener, a consumer.send(msg) instruction. It will respond to the sender.
- **b** All other choices are true.
- **c** If a single receiver is started in each execution of this pair of programs, the argument for its consumer.connect() call may be tcp://*:8888.
- **d** We may start, in any order, multiple receivers and a single sender. The resulting set of processes will not generate any error and messages will be eventually delivered.

25 *The ØMQ REQ-REP communication pattern is considered synchronous because*

- **a** The reception of request and response messages cannot be handled using listeners in the server and client processes, respectively.
- **b** A REQ socket cannot send and transmit two consecutive request messages if there is no intermediate response message delivery between those sendings.
- **c** It's unable to provide communication persistency
- **d** A server cannot handle concurrent connections with more than one client.

26 *Let us consider the following pair of Node.js programs that use 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')
})
```

All instances of those programs run in the same computer. Choose the true statement:

- **a** No error arises if we start and simultaneously run two server processes.
- **b** If we eventually start a server process, multiple client processes may be started. Those clients will eventually receive a response.
- **c** A server may receive and deliver a request message sent by a client before sending the response to a previously delivered request from another client.
- **d** All other choices are true.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A

DNI NIE PASSPORT

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

ETSINF - TSR

First Partial - 02/11/2023

Surnames ....

Name ....

Mark like this Like this do not mark

NOT ERASING, correct with concealer

First Partial

	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				