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 *Some mechanisms that improve the Wikipedia throughput are:*

- **a** Replication
- **b** Load balancing and caching, with reverse proxies
- **c** All other choices are correct.
- **d** Persistent data partitioning (i.e., sharding)

2 *The theoretical computing model based on guards and actions…:*

- **a** …is directly related to event-oriented programming.
- **b** …may only be applied to messaging systems.
- **c** …is only related to multi-threaded programming.
- **d** …only makes sense for single-process applications.

3 *Some relevant aspects of distributed systems are:*

- **a** Fault tolerance.
- **b** Shared usage of several resources.
- **c** All other choices are correct.
- **d** Coordination of actions from different components.

4 *Which of these elements is related to LAMP?*

- **a** A messaging middleware.
- **b** A type of application container.
- **c** A Node.js module.
- **d** Wikipedia.

5 *In regard to cooperative computing:*

- **a** Wikipedia is an example.
- **b** It has all characteristics mentioned in the other choices.
- **c** It is mainly used in peer-to-peer systems.
- **d** It is adequate for solving some scientific and engineering problems that may be partitioned into other smaller ones.

6 *In cloud computing…:*

- **a** SaaS providers supply software services to their users.
- **b** IaaS providers supply virtual machines in order to run some deployed services.
- **c** PaaS providers supply a development environment with some operating system characteristics.
- **d** All other choices are correct.

7 *Let us consider this Node.js program, First fragment, that uses the events module. When that program runs, it shows: Event print: 1 times*

```
// First fragment
const ev = require('events')
const emitter = new ev.EventEmitter()
const e1 = "print"
let num1 = 0// Event counter
emitter.on(e1, function() {
  console.log("Event " + e1 + ": " + ++num1 + " times.")
})
emitter.emit(e1)
```

What is shown on the screen when that program is modified, generating this Second fragment (where we have only moved the original last line)?

```
// Second fragment
const ev = require('events')
const emitter = new ev.EventEmitter()
const e1 = "print"
let num1 = 0// Event counter
emitter.emit(e1)  // emit() before on().
emitter.on(e1, function() {
  console.log("Event " + e1 + ": " + ++num1 + " times.")
})
```

a No other choice is correct.

b It shows the same message than in the original program: Event print: 1 times.

c No message is shown.

d An exception is raised in emitter.emit(e1).

8 *These two Node.js programs use the net module. How many clients, connected simultaneously, may be handled by 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')
   // Send my PID.
   client.write(process.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 One. This means that it may only handle a new client when the current one disconnects.

b Two.

c None.

d The number is not limited.

9 *Let us assume that we run this Node.js code snippet:*

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

a Function f has been incorrectly declared, since argument cb has no type.

b The snippet runs entirely in a single turn of the event loop (i.e. turn queue).

c No other choice is true.

d Its second line calls function f, with a callback that runs asynchronously.

10 *About module* `net` *in Node.js:*

a It provides TCP sockets that may send or receive data through the network. With them, the write() method sends data and the read() method receives data. Write() returns the number of propagated bytes, while read() returns an array with the read data.

b No other choice is true.

c In order to create a socket, its socket type should be specified. Among others, it may be req or rep.

d Since TCP sockets are instances of EventEmitter, one process may run socket.emit(event) and that event will be received in the other side of the connection with socket.on(event, callback).

11 *Let us consider this Node.js program:*

```
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 When it runs, it shows several lines on the screen, always the same set, but the order of those lines may change in each execution, since the program is asynchronous.

b The program has some error that breaks its execution. Concretely, a call to 'genera_cb()' cannot be used as an event listener.

c If the program runs several times, this output may be provided in some of them:
e0:1
e1:1
e0:2

d When it runs, it always shows this result:
e0:1
e0:2
e1:1

12 *How many arguments should use the function that handles an event occurrence?*

a Two: the name of the event and the name of the own function.

b One: the name of the event.

c That number depends on the number of arguments used in the corresponding emit() call.

d None: those callbacks cannot have any formal parameter.

13 *Which is the sequence of numbers shown when this program runs?*

```
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 2 3 1

b 1 2 3

c 2 1 3

d 1 3 2

14 *Which is the sequence of numbers shown on the screen when this program runs?*

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

a 3 3 3

b 0 1 2

c None, since an error is generated.

d 2 1 0

15 *What is written on the screen when this program runs?*

```
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)))
```

a An error is shown and, additionally, a number that starts with '5'.

b Seven characters in the first line and a number than ends with '5' in the second line.

c Seven characters in the first line and an error in the second line.

d Only an error message is shown.

16 *Let us consider these two programs: client.js and server.js. Let us also consider that in all executions only one client and one server start.*

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

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

a The start order of both processes does not matter. In all cases, the result is the same. The client's screen shows HOLA and the client ends.

b If we start first the server and, afterward, the client, then we read HOLA on the client's screen and the client ends soon thereafter. If we start again the client, this second client execution does not show anything on the screen.

c The programs are defective, since client calls bind() while server calls connect(). They should be called the other way round.

d If we start first the server and, afterward, the client, then we read HOLA on the client's screen and the client ends soon thereafter. The client ends because it has closed its socket, and this also causes server termination.

17 *Choose the FALSE statement about ZeroMQ:*

a ZeroMQ may use as its transport TCP or IPC (Interprocess communication).

b ZeroMQ provides strong persistency. Therefore, messages sent by some sender process are stored on disk until they are delivered to the intended receiver.

c REQ/REP sockets are asynchronous and they implement a synchronous request/reply communication pattern.

d ZeroMQ implements 'framing', an aspect that is not considered in TCP/IP. Concretely, module 'net' in Node.js does not provide 'framing'.

18 *Let us consider these programs:*

```
// client.js
// usage: 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++) {
  ... // other actions
  req.send([id,msg[i]])
}
req.on('message',
 (c,id,m)=>console.log(c+':'+id+':'+m))
```

```
// server.js
// usage: 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)=>{
  ... // other actions
  rep.send([c,id,m])
}
```

If we simultaneously start these commands in different terminals:

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

a No other choice is true.

b Independently on the time needed to run the other actions (…) in client and server, in the outgoing queue of C1 there may not be more than an pending message.

c Independently on the time needed to run the other actions (…) in client and server, in the incoming queue of S1 there may be more than one pending message.

d In the incoming queue of S1 there may simultaneously be messages from C1 and C2.

19 *Let us consider these programs:*

```
// client.js
// usage: 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++) {
  ... // other actions
  req.send([id,msg[i]])
}
req.on('message',
 (c,id,m)=>console.log(c+':'+id+':'+m))
```

```
// server.js
// usage: 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)=>{
  ... // other actions
  rep.send([c,id,m])
}
```

If we simultaneously start these commands in different terminals:

```
node client C 2 8000 8001 A B C D E
node server S1 8000
node server S2 8001
```

a No other choice is true.

b Independently on the time needed to run the other actions (…) in client and server, messages are processed in order A B C D E.

c The order in which those requests are processed depends on the duration of the other actions (…) in client and server.

d Independently on the time needed to run the other actions (…) in client and server, messages are processed in order A D B E C.

20 *Choose the correct statement:*

- **a** A given process may define a PULL socket and a PUSH socket.
- **b** A PULL socket cannot receive connections from more than one PUSH socket.
- **c** A PUSH socket does not have any outgoing queue.
- **d** A PUSH socket cannot receive connections from more than one PULL socket.

21 *Choose the correct statement about ZeroMQ:*

- **a** Messages sent by a PUB socket are distributed among its subscribers using a Round-Robin strategy.
- **b** A subscriber cannot select the type of messages that it receives from the publisher.
- **c** The subscribe() operation in a PUB socket states to which other PUB socket it connects to.
- **d** A subscriber that starts once the publisher has already been started may lose part of the sent messages.

22 *In regard to the bind() and connect() operations in ZeroMQ, choose the true statement:*

- **a** No other choice is true.
- **b** We cannot use multiple connect() calls onto a URL that has only received a single bind() call.
- **c** If a connect() is attempted and the other side has not called bind(), the program aborts.
- **d** If a bind() is attempted and the other side has not called connect(), the program aborts.

23 *In ZeroMQ, may we broadcast messages from a single sender process to two receiver processes without using PUB/SUB sockets?*

- **a** Yes: we may use an array of REQ sockets (one per subscriber) instead of PUB and REP instead of SUB. Each pub.send(m) is replaced with req[i].send(m) onto each socket in that array.
- **b** Yes: we may use an array of PUSH sockets (one per subscriber) instead of PUB and PULL instead of SUB. Each pub.send(m) is replaced with push[i].send(m) onto each socket in that array.
- **c** Yes: we may use PUSH instead of PUB and PULL instead of SUB. Each pub.send(m) is replaced with one push.send(m).
- **d** No.

24 *In regard to message filtering at the subscriber side in the PUB/SUB ZeroMQ communication pattern:*

- **a** That filtering is not possible. Subscribers receive all sent messages.
- **b** The publisher may choose using subscribe() the type of messages to be received by each subscriber.
- **c** The subscriber may choose using subscribe() the type of messages it wants to subscribe to.
- **d** The subscriber may specify the type of messages it subscribes to connecting to the URL used in sub.connect().

25 *Choose which is the sequence of numbers (one per line) shown by this program:*

```
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 None, since it generates an error.

c 0 0 1 2 3

d An increasing sequence 0 1 2 … that only ends if we abort the program with Ctrl+C.

26 *Choose which is the sequence of numbers (one per line) shown by this program:*

```
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 None, since it generates an error.

d 2 9 2 8

27 *Choose what this program writes on screen:*

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

a Nothing, since it generates an error.

b [false, false, false, false]

c [true, true, true, true]

d [false, true, false, true]

28 *In the emisor3 problem handled in the second session of Lab 1, the bulletin recommends a call to setTimeout() in order to start the first stage. Choose which of these statements is true:*

a Function etapa() should not call again setTimeout(), since setTimeout() has already been called. Instead, we should recursively call etapa(), without calling setTimeout() again.

b No other choice is true.

c We may replace setTimeout() with setInterval(). Since setInterval() programs the timer in a cyclic way, no other call to setInterval() or setTimeout() will be needed in etapa().

d A call to setTimeout() in etapa() implies multiple recursive calls that may overflow the maximum process stack size.

29 *In regard to the netClientLoad-netServerLoad problem handed in the labs, where the provided function getLoad() returns the workload in the local computer, choose the true statement:*

a Program netClientLoad should contain in its code the function getLoad().

b None of those two programs should contain function getLoad().

c Both programs, netClientLoad and netServerLoad, should contain in its code the function getLoad().

d Program netServerLoad should contain in its code the function getLoad().

30 *The programmable proxy (ProxyProg) used in the labs may only be used with HTTP clients because…*

a ProxyProg is very basic and it only understands the HTTP protocol. Thus, no other protocol or HTTP variant may be handled by ProxyProg.

b Unlike the configurable proxy (ProxyConf), ProxyProg receives [IP, port] pairs that are only useful in a URL when its target is a web server.

c Our assumption is false: ProxyProg correctly manages HTTP because it works at a lower network-related layer.

d In this course, we have not studied how to build a client process that sends requests at a layer different from HTTP.

31 *Let us consider this program:*

```
// 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 listening on " +
"port: " + LOCAL_PORT);
```

In the original code of proxy.js (Lab 1), that uses other names for those sockets, the connect() callback includes some statements that have this goal:

a Copy the messages received from s2 to socket s1 and vice versa.

b Create a socket in order to send the requests received from s1, sending back the response to clients using s2.

c Handle the 'message' event on each socket, processing messages in JSON format, and forwarding them from one socket to the other.

d Create a socket in order to send the requests received from s2, sending back the response to clients using s1.

32 *In Lab 1, let us consider the following extension of the original proxy.js program, that tries to implement the programmable proxy variant:*

```
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);
```

Which of the following efects arises when this proxy receives a programming message:

a This proxy aborts due to an error, since it cannot modify the parameters related to the remote server.

b The existing connections are broken.

c The existing connections are redirected onto the new programmed server.

d Subsequent connections are forwarded to the new programmed server, but the existing ones are not disturbed.

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					