TSR: First Partial

This exam consists of 20 multiple choice questions. In every case only one answer is correct. You should answer in a separate sheet. If correctly answered, they contribute 0,5 points to the exam grade. If incorrectly answered, the contribution is negative: -0.167. So, think carefully your answers.

THEORY

1. Regarding Unit 1, these sentences correctly describe some aspects of distributed systems:

a	Every concurrent system is a distributed system. False. All distributed systems or applications are examples of concurrent systems, but not all concurrent systems are distributed. For instance, a Java program that uses two threads of execution (e.g., in order to interact with the user and to access data files) generates a concurrent process, but that process is not a distributed application.
b	The e-mail service is an example of distributed system. True. In an e-mail service there are different types of agents (e-mail server, e-mail reader...) and those agents are usually deployed onto multiples computers and need to interact among them.
c	The agents in a distributed system cannot share any resource since each agent must be placed in a different computer. False. Some resources may be shared by those agents; otherwise, no mutual exclusion algorithm (and some examples of them have been presented in Seminar 2) would have been needed in a distributed system.
d	The developer of a distributed application does not need to worry about fault tolerance, since it is inherently guaranteed by the distributed system. False. In order to provide fault tolerance, component replication is needed. Such replication is not automatically provided by all kinds of "distributed systems". Replication management is commonly a responsibility of the distributed application (although some specific middleware layers may help in that management).

2. One of the reasons for stating that Wikipedia is a scalable distributed application is…

a	Since its first release, it has been implemented following the SaaS cloud service model. False. The first releases of Wikipedia were generated in 2001, before the SaaS cloud service model was popular. Beside this, at that ime, the Wikipedia project was not yet widely known and it could run using a few computers at its server side.
b	It is a LAMP system, and all systems of this kind are highly scalable. False. A LAMP architecture is not necessarily highly scalable. It should be complemented with an adequate caching strategy and an efficient component management in order to be scalable.
c	It uses a P2P interaction approach and this strongly enhances its scalability. False. It uses a client-server interaction approach.
d	It uses caching at its reverse proxies and component replication. True. Both caching and replication improve its scalability.

3. The main goal of the PaaS cloud service model is…

a	To automate the configuration, deployment and upgrading of distributed services and their reconfiguration under varying loads. True. These are its main goals.

b	To automate infrastructure provision. False. That is the main goal of IaaS systems.
c	To provide distributed services under a pay-as-you-go model. False. That is the main goal of SaaS systems.
d	To provide persistent data under a pay-as-you-go model. False. That is the main goal of DBaaS systems (a subset of the IaaS model).

4. A simple distributed system model was proposed in Unit 2 because that model…

a	…ensures data persistence. False. System models (and models in general) cannot ensure any functional property. In order to ensure data persistence (i.e., durability and fault tolerance) data elements must be replicated.
b	…is needed for comparing multi-threaded and asynchronous programming. False. That comparison is not the focus of the system model explained in Unit 2.
c	…facilitates a good basis for designing distributed algorithms and protocols and for reasoning about their correctness before starting their implementation. True. That is the main goal of all distributed system models.
d	…shows that activity blocking prevents services from scaling. False. The proposed model is unable to directly show those facts at its high level of abstraction. It provides a basis for analysing algorithms, but the model itself cannot show those properties (blocking behaviour, scalability...). Such properties belong to specific algorithms able to provide them.

5. This is the best solution for ensuring data persistence:

a	Usage of stateful servers. False. "Stateful servers" refer to servers that are able to maintain all context information about their interactions with each client. This doesn't guarantee data persistence, since if any stateful server crashes its clients cannot interact with other new servers that replace the failed one: its session context information is lost. In order to prevent such loss from happening, a careful (and expensive regarding both resources and time) replication strategy is needed.
b	Data replication. True. Data should be replicated in order to survive to failures and guarantee their durability and accessibility.
c	Use the most reliable hard disk drives. False. Event he most reliable storing media will eventually fail and lose its data.
d	Avoidance of concurrent accesses to data. False. Concurrent accesses may endanger data consistency, but not data persistence.

6. The simple system model described in Unit 2 is directly supported by the asynchronous programming paradigm because...

a	…asynchronous programming is based on causal communication. False. Asynchronous programming doesn't depend on any type of communication order.

b	…processes in that simple system model are multi-threaded. False. Asynchronous programming doesn't imply multi-threading. Indeed, multi-threaded programming is usually synchronous.
c	…there is a direct translation between guards + actions in the model and events + callbacks in asynchronous programming. True. In the model, the actions to be executed by processes are protected by preconditions or guards. In the aysnchronous programming paradigm, a program executes fragments of code when an event happens. Therefore, such event behaves as a precondition or guard and, once satisfied, starts the execution of a callback (that corresponds to the action code in the model).
d	…processes inherently follow the stop failure model in the asynchronous programming paradigm. False. The asynchronous programming paradigm doesn't imply or depend on any failure model.

7. A messaging middleware layer (MML) is more convenient than remote method invocation (RMI) for building scalable applications because…

a	MML provides location transparency and RMI cannot do this. False. RMI may provide location transparency (for instance, using proxies) while MML (as it has been used in the ZeroMQ example) not always provides location transparency.
b	MML is inherently asynchronous, while RMI is synchronous. True. ZeroMQ is an example of MML and provides asynchronous communication. A process may continue with its execution once it has called socket.send(), even in the REQ-REP communication pattern. On the other hand, remote invocations in RMI block the caller until a reply is received. Therefore, MML is asynchronous and RMI is synchronous.
c	Processes that use MML assume a shared resource space. In RMI none of the processes shares any resource. False. MML provides a non-shared image (due to its lack of location transparency) while RMI usually provides a system image where all resources are potentially usable as local entities (due to its location transparency).
d	Processes that use MML are automatically replicated. In RMI, replication isn't allowed. False. MML and RMI are not directly related with replication management. None of them automatically replicates agents or resources (nor prevents their replication from being possible).

8. Persistent messaging…

a	…implies location transparency. False. As it has been discussed in the previous question, messaging middleware doesn't provide location transparency.

b	…is automatically guaranteed when a naming middleware is used. False. Persistency in message-based communication doesn't depend on any naming service. Persistency depends on the capacity of message brokers or intermediaries for holding messages in transit while the other communication side is not ready yet.
c	…may be easily implemented by intermediate message brokers. True. They should hold the messages until they can be delivered to their target processes.
d	…cannot be used in asynchronous communication. False. ZeroMQ successfully combines asynchronous communication with partial persistency.

SEMINARS

9. Considering this program:

```
var fs=require('fs');
if (process.argv.length<5) {
  console.error('More file names are needed!!');
  process.exit();
}
var files = process.argv.slice(2);
var i=-1;
do {
  i++;
  fs.readFile(files[i], 'utf-8', function(err,data) {
      if (err) console.log(err);
      else console.log('File '+files[i]+': '+data.length+' bytes.');
  })
} while (i<files.length-1);
console.log('We have processed '+files.length+' files.');
```

These sentences are true if we assume that no error aborts this program execution:

a	This JavaScript program prints in all iterations, among other data, the name of the last file provided in the command line. True. This program is an example of loop that in each iteration uses an asynchronous callback. In those cases, the callback cannot rely on the value of the iteration counter. Note that those callbacks, being asynchronous, are usually executed once the loop has finished. Because of this, in the regular case, all callbacks will print the name of the same file: the last one.

b	It prints the name and length for each one of the files received from the command line. False. In order to provide such output a closure is needed. Without closure, the lengths are correct, but the names aren't. The name of the last file is printed in all cases.
c	It prints "We have processed N files" at the end of its execution, being N the number of file names given as arguments. False. Although this message is printed in the last sentence of the program, that is the last sentence of the first execution turn and the message is printed before starting the execution of callbacks. Therefore, it is the first message being printed (instead of the last one).
d	It discards the names of the first two files given as arguments to this program. False. None of the file names is discarded. Line 6 (var files = process.argv.slice(2);) discards the word "node" and the name of the program being executed, but not the file names given as arguments.

10. Regarding the program shown in the previous question...

a	It needs multiple turns for completing its execution, since each file being read requires its own turn. True. Each file is read using an asynchronous callback. Asynchronous callbacks are executed in separate turns.
b	It generates an exception and crashes if any error happens when it tries to read a file. False. If any error happens in those cases, it is reported in the first argument of the callback. The callback code manages that parameter printing information about the error, without rising any exception or aborting the process.
c	This program is incorrect. It must use "var i=0" to initialise variable "i" in order to be correct. False. The initial value of that variable is correct. Note that arrays are indexed starting at zero by default. Since the do{...}while() loop increases the value of "i" in its first statement, it should be initialised to -1 in order to correctly access the slots of the "files" array.
d	It always prints the same length in all iterations. We need a closure in order to avoid this faulty behaviour. False. The length of the file is correctly shown in all iterations, since it depends on the "data" parameter and its value is given at callback invocation time. A closure is needed for remembering the file name, but not for managing the file length.

11. Regarding the mutual exclusion algorithms seen in Seminar 2, it is true that...

a	The central server algorithm minimises the amount of messages being needed. True. Among all mutual exclusion algorithms described in Seminar 2, that based on a central server is the one that needs the minimal amount of messages. It only requires that each process interacts with the central server, using point-to-point communication (1 message per request and 1 per reply), while the other algorithms were based on multicasting (sending N or N-1 messages in several of their steps) or on a logical ring (continuously forwarding the token when no requestor exists).

b	The virtual unidirectional ring algorithm has a synchronisation delay of 1 message. False. Its synchronisation delay is 1 message in the best case, but may be N-1 messages in the worst case.
c	The synchronisation delay of the multicast algorithm with logical clocks is 2N-2 messages. False. Its synchronisation delay is one message.
d	The multicast algorithm based on quorums (i.e., its version described in the slides) complies with all 3 mutual exclusion correctness conditions. False. It is prone to deadlocks and, due to this, it cannot ensure the liveness condition of these algorithms.

12. Considering this program...

```
var ev = require('events');
var emitter = new ev.EventEmitter;
var num1 = 0;
var num2 = 0;
function emit_e1() { emitter.emit("e1") }
function emit_e2() { emitter.emit("e2") }
emitter.on("e1", function() {
   console.log( "Event e1 has happened " + ++num1 + " times.")});
emitter.on("e2", function() {
   console.log( "Event e2 has happened " + ++num2 + " times.")});
emitter.on("e1", function() {
   setTimeout( emit_e2, 3000 )});
emitter.on("e2", function() {
   setTimeout( emit_e2, 2000 )});
setTimeout( emit_e1, 2000 );
```

The following sentences are true:

a	Event “e1” happens only once, 2 seconds after this program is started. True. The last statement of this program sets that event “e1” should be rised in 2 seconds. That event has two listeners: one prints how many times has happened e1, and the other sets that event “e2” should be rised in 3 seconds. None of the listeners for event “e2” will rise again “e1”. Therefore, “e1” has only happened once, 2 seconds after the program start.
b	Event “e2” never happens. False. Event “e2” happens multiple times. The first one, 5 seconds after the program start. The remaining ones are periodical, with an inter-event interval of 2 seconds.
c	The period of “e2” is five seconds. False. Its period is 2 seconds.
d	The period of “e1” is three seconds. False. Event “e1” only happens once.

13. Considering the program shown in the previous question...

a	The first “e2” event happens five seconds after the program is started. True. This has been explained in the previous question.

b	No event is generated in its execution, since its emit() calls are incorrect. False. Events are generated correctly.
c	We cannot have more than one listener for each event. Therefore, the program is aborted by an exception in its third emitter.on() call. False. Events may have multiple listeners.
d	None of its events happens periodically. False. Event "e2" happens every 2 seconds.

14. The ØMQ REQ-REP communication pattern is considered synchronous because...

a	It follows the client/server interaction pattern and in that pattern the client remains blocked until a reply is received. False. The sender of a request message using a REQ socket may continue with its execution once that message has been sent. Replies to that message are managed using a 'message' listener, asynchronously.
b	Both REQ and REP sockets are bidirectional; i.e., both may send and receive messages. False. Although those sockets are bidirectional, bidirectionality doesn't imply synchrony.
c	The output queue in the REQ socket has a limited capacity. It may only hold one message. False. Output queues in all ZeroMQ sockets that allow message sending have a larger capacity. They are not limited to a single message.
d	REQ sockets cannot transmit a request until the reply to its previous request is received. REP sockets cannot forward a reply until its request is received. True. This is the characteristic that introduces some degree of synchrony in the REQ-REP communication pattern.

15. Considering these two node.js programs...

```
// server.js
var net = require('net');
var server = net.createServer(
 function(c) {//'connection' listener
  console.log('server connected');
  c.on('end', function() {
   console.log('server disconnected');
  });
  c.on('data', function(data) {
   console.log('Request: ' +data);
   c.write(data+ 'World!');
  });
});
server.listen(9000);
```

```
// client.js
var net = require('net'); var i=0;
var client = net.connect({port:
  9000}, function() {
    client.write('Hello ');
  });
client.on('data', function(data) {
  console.log('Reply: '+data);
  i++; if (i==2) client.end();
});
client.on('end', function() {
  console.log('client ' +
    'disconnected');
});
```

The following sentences are true:

a	The server terminates after sending its first reply to the first client. False. The server code has no statement forcing its termination (e.g., a process.exit() call). Note that when a connection is ended it only prints a message reporting this fact. Because of this, it is able to handle multiple connections, replying to each incoming message with another message that appends the 'World!' string to the contents of the incoming request.

b	The client never terminates. True. It never closes its open connection, since it would close it once a second message is received, but only a reply is sent by the server. This expected second message never arrives. Therefore, the client remains indefinitely expecting that second message.
c	This server can only handle one connection. False. Servers that use the "net" module handle multiple connections.
d	This client cannot connect to this server. False. Both client and server use the same port number. They may communicate without problems when they have been started at the same computer.

16. Leader election algorithms (from Seminar 2)...

a	...have no safety condition. False. In order to be correct, all distributed algorithms should respect safety and liveness conditions (at least one condition of each type). The safety condition for leader election algorithms is that there cannot be more than one simultaneous leader in the system.
b	...may be infinitely looking for a leader process. False. The general liveness property for this kind of algorithms requires that eventually a leader is chosen. Therefore, the election cannot require an infinite interval to conclude.
c	...must ensure that a single leader is chosen. True. This is its safecty condition.
d	...must respect causal order. False. That is not a requirement in these algorithms. It has been presented as a correctness condition for mutual exclusion algorithms, but not for leader election.

17. We want to implement a mutual exclusion service using NodeJS and ØMQ, using the first algorithm explained in Seminar 2: the central server algorithm. In order to implement this service, the best of the following options is...

a	The server needs a DEALER and a ROUTER socket to balance the load among its clients. False. In that algorithm there is a set of general processes that may require their access to the critical section. Those requests follow, in a general way, a request-response pattern in the CS-entry protocol and a unidirectional sending in the CS-exit protocol. The ROUTER-DEALER communication pattern is unable to manage this. Those two sockets are used by broker processes in order to forward messages to a (potentially large) set of servers or workers. That pattern of interaction isn't needed in that mutual exclusion algorithm.

b	Each client needs a DEALER socket to interact with the server. True. DEALER sockets may implement both the request-response pattern in the CS-entry protocol and the unidirectional sending in the CS-exit protocol.
c	Each client needs a REP socket to interact with the server. False. REP sockets cannot be used for initiating the CS-entry or the CS-exit protocols at the requestor side. Note that a REP socket cannot start communication until a request message has been received by that socket.
d	Each client needs a SUB socket to interact with the server. False. SUB sockets may only receive messages. They cannot be used for sending messages. Therefore, they cannot implement the CS-entry or the CS-exit protocols at the requestor side.

18. We want to implement a mutual exclusion service using NodeJS and ØMQ, using the second algorithm explained in Seminar 2: the (virtual) unidirectional ring algorithm. In order to implement this service...:

a	We need to use any leader election algorithm in order to choose a coordinator process. False. The ring algorithm is symmetrical. All processes behave in the same way. Therefore, no coordinator is needed in that algorithm.
b	All processes have the same role and need a REP socket to send messages and a REQ socket to receive them. False. REP and REQ sockets are bidirectional and follow a synchronous interaction pattern: a process cannot send two consecutive messages without any interleaving reception. Therefore, they cannot implement the unidirectional pattern being required in this algorithm.
c	All processes have the same role and need a PUSH socket to send the token and a PULL socket to receive it. True. These sockets are unidirectional and comply with the communication requirements of this algorithm.
d	All processes have the same role and use a PUB socket to send the token and a DEALER socket to receive it. False. Although a PUB socket may be used for sending messages and a DEALER for receiving them, the PUB socket uses a broadcasting approach instead of a point-to-point one. Thus, those sockets are not appropriate for implementing this algorithm.

19. Considering these programs...

```
//client.js
var zmq=require('zmq');
var rq=zmq.socket('dealer');
rq.connect('tcp://127.0.0.1:8888');
for (var i=1; i<100; i++) {
  rq.send(''+i);
  console.log("Sending %d",i);
}
rq.on('message',function(req,rep){
  console.log("%s %s",req,rep);
});
```

```
// server.js
var zmq = require('zmq');
var rp = zmq.socket('dealer');
rp.bindSync('tcp://127.0.0.1:8888');
rp.on('message', function(msg) {
  var j = parseInt(msg);
  rp.send([msg,(j*3).toString()]);
});
```

The following sentences are true:

a	Both client and server exchange messages in a synchronous way in this example, since they follow a request-reply pattern. False. Both client and server use DEALER sockets and none of them remains blocked (once it has sent a message) until another message is received. Therefore, this example doesn't use synchronous communication. Indeed, in this example, the client may send all its 100 messages before their first reply will be received.
b	The server returns a message with 2 segments to the client. The second segment contains a value that is 3 times greater than that in the first segment. True. That is the functionality of the 'message' event listener in the server.
c	Client and server may be run in different computers. They interact without problems in that case. False. Both use 127.0.0.1 as their bind() or connect() IP address. This means that both interact with another process in the local computer. Therefore, they cannot be placed in two different computers.
d	No message is sent by the client since the `''+i` statement generates an exception and the program aborts at that point. False. The goal of that statement is casting the value of 'i' to a string.

20. Please consider which of the following variations will generate new programs with the same behaviour as that shown in question 19 (A --> B means that statement A must be replaced by statement B)...

a	The 'rq' socket should be of type 'PULL' and the 'rp' of type 'PUSH'. False. The sockets to be used must support bidirectional communication. PUSH and PULL sockets are unidirectional.
b	The 'rq' socket should be a 'PUSH' socket and 'rp' should be a 'PULL'. False. The sockets to be used must support bidirectional communication. PUSH and PULL sockets are unidirectional.
c	Client: rq.connect('tcp://127.0.0.1:8888'); --> rq.bindSync('tcp://*:8888'); Server: rp.bindSync('tcp://127.0.0.1:8888'); --> rp.connect('tcp://127.0.0.1:8888'); True. This is the best option among all those presented in this question. The behaviour will be exactly the same if we assume that a single client and a single server need to be started.
d	The 'rq' socket should be of type 'REP' and 'rp' should be of type 'REQ'. False. A REP socket cannot be used to start the interaction with another process. Therefore, a client cannot use a REP socket in order to interact with a server process.