TSR / NIST – First Partial (Retake)

This exam consists of 10 multiple choice questions. In every case only one answer is correct. You should answer in a separate sheet. If correctly answered, they contribute 1 point to the exam grade. If incorrectly answered, the contribution is negative: -0.333. So, think carefully your answers.

THEORY

1. In which cloud service model does the application developer belong to the cloud provider company?

X	SaaS True. In this case the service being provided is an elastic software service. Therefore, one or more distributed applications must be developed by that provider company. Thus, in this case software developers belong to the provider company.
	PaaS False. In this case the service provider offers a platform that automates some development tasks and the application deployment as its main service. The application developer uses those services. Thus, the developer belongs to the customer company in this case.
	IaaS False. In this case the service provider offers virtual infrastructure elements as its main service. The application developer uses that infrastructure. Thus, the developer belongs to a customer company in this case.
	None False. See the explanation in case “c”.

2. Which of these statements about the Wikipedia system IS FALSE?

X	Most of its components are written in NodeJS. False. At the moment, those components are not developed in NodeJS. See the explanation in part “d”.
	It uses reverse proxies. True. In order to increase its performance, multiple reverse proxy instances are used in the Wikipedia system.
	It uses component replication. True. Many of its components are replicated and there are many instances of them.
	It is an example of LAMP system. True. MediaWiki, the core of the Wikipedia system, follows the main characteristics of a LAMP system, since it uses: Linux nodes for its deployment (L), Apache as its web server (A), MySQL (or any compatible relational DBMS) as its persistency layer (M) and PHP as its scripting language in order to develop many parts of its logic (P).

TSR / NIST – First Partial (Retake)

3. Which IS NOT an advantage of asynchronous servers?

X	They are multi-threaded. Thus, their activity is not blocked when one of their threads gets blocked. False. In the common case, they are single-threaded, as we have explained above.
	They do not cause memory sharing problems. True. Since there is a single thread of execution per process, there cannot be memory sharing problems among the activities to be run in those processes.
	Their execution is usually fast: short event handlers that may be run sequentially, without many context switches. True. Activities are started when any new event occurs. Those activities are short in the common case and they may be executed in a single CPU share.
	In many cases, they do not need any internal concurrency control mechanism. True. Many asynchronous servers are written in JavaScript or similar event-oriented programming languages. In those languages, there is a single thread of execution per process. This eliminates the need of concurrency control mechanisms, since there is no concurrency in the execution of those processes.

4. Message-based communication is more asynchronous than object invocation because...

X	…process interactions do not need to follow a request-reply pattern. True. Object invocation and RPC mechanisms always follow a request-reply pattern. This means that the sender of a request commonly becomes blocked waiting for a response, implementing in this way a type of synchronous communication. On the other hand, message-based communication is only focused on "send()" and "receive()" actions, without any additional semantics. Therefore, once a message is sent, its sender may go on with no restriction.
	…the messages used in the former are smaller than the requests and replies in the latter. False. Both communication mechanisms do not enforce any constraint on message size. This implies that the messages used in message-based communication do not need to be smaller than those used in object invocations.
	…object invocation relies on client caches. False. In the regular case, object invocation mechanisms do not require the usage of client caches. Besides, if an object invocation middleware used client caches it would be more asynchronous (or free) than any other system that doesn't use those caches. The usage of caches allows a shorter blocking interval than when no cache is used.
	…object invocation uses reverse proxies. False. In the regular case, object invocation mechanisms do not require the usage of reverse proxies. Besides, if an object invocation middleware used reverse proxies it would be more asynchronous (or free) than any other system that doesn't use those proxies since reverse proxies may cache invocation results. The usage of reverse proxies allows a shorter blocking interval than when no cache is used.

TSR / NIST – First Partial (Retake)

SEMINARS

5. We have written this NodeJS program:

```
const fs = require('fs');
process.stdin.resume();
process.stdin.setEncoding('utf8');
process.stdin.on('data',function(name) {
      var filename = name.slice(0,name.length-1);
      fs.readFile(filename,'utf8', function(err,contents) {
            if (!err) console.log(contents);
            else console.log(err);
            console.log("");
            console.log("File name:");
      })
})
console.log("File name:");
```

The goal of that program is…

X	Implement an infinite loop. In each iteration, a string is read and, if a file with that name exists, its content is shown on the screen; otherwise an error is shown. True. This program is continuously reading the standard input; i.e., the keyboard. Each time a new-line character is entered from the keyboard, the current input line is delivered to the "data" event listener whose code starts in line 4 of this program. That block of code removes the last read character (i.e., the "new-line" one). The remaining part of that introduced string is taken as a file name. That file name is used as the first argument in a fs.readFile() call, specifying 'utf8' encoding for the result of that read action. The text contents of that file, if it exists, are passed in the "contents" parameter of the readFile() callback. When no error is detected, those contents are shown on the screen. Otherwise, the error message is shown. Once this has been done, an empty line is printed and the "File name:" prompt is shown again, asking for another file name. As a result of this behaviour, an infinite loop is implemented.
	Read a string from the standard input and look for that string in the contents of a file called "filename". False. See the explanation in option "d".
	Read a string from the standard input and show that string in the screen if a file with that name exists. False. See the explanation in option "d".
	Show the content of a file called "data" that is placed in the current folder. Once this is done, a "File name:" string is shown on the screen. False. See the explanation in option "d".

TSR / NIST – First Partial (Retake)

6. Considering this NodeJS program:

```
var net = require('net');
net.createServer(
  function(c) {
    c.on('data', function(msg) {
      c.write(msg+msg);
    })
  }
).listen(9000);
```

Once started, that program does the following:

X	It listens for incoming connections on port 9000. Each received message is sent back to its sender, duplicating its content. True. The net.createServer() function uses a connection handler as its single argument. In this example, we have used an anonymous function in that place. This anonymous function uses a unique parameter (the socket that has been generated when a connection has been set by a remote client). In the body of that anonymous function we have a 'data' event handler. Such event handler uses one parameter. Each time it is invoked, the received message (from the client) is passed as that argument. The body of the function uses the "write()" method on that socket in order to send a reply to the client. That reply consists in the concatenation of the message to itself.
	It raises an exception, since variable "c" has not been declared. False. No exception is generated due to this. Variable "c" is declared as a parameter in a function. No "var" or "const" keyword is needed in that case. Therefore, that variable has been used in a correct way in this program.
	It raises an exception, since we cannot use the "listen" operation on something that is a function, instead of a 'net' socket. False. The result of a net.createServer() invocation is a socket. We use the listen operation on that socket.
	Nothing, since there is no useful statement in that short program. No error or exception is reported. False. See explanation in part "c".

7. Someone has said: "The mutual exclusion algorithm with a central server shown in Seminar 2 CANNOT BE implemented in NodeJS and ZeroMQ". Which of the following alternatives is valid about that sentence?

X	It's false. The algorithm can be implemented (e.g., using a ROUTER socket in the central server and a REQ in each other process) and it makes sense. Yes. That is a possible basic implementation for that algorithm.

	It's true, since the server needs to send a message to exactly two processes in one of the algorithm steps, and such action is not supported by ZeroMQ sockets. No. The algorithm doesn't need to send any of its messages to two processes at the same time. All inter-process interactions are made between one (and only one) client and the server.
	It's false. The algorithm can be implemented in that framework. However, its implementation is nonsense, since NodeJS processes cannot share any resource. No. Even in distributed deployments that use a single process per computer, those participating processes may share resources (e.g., the access to a multi-ported external device).
	It's true, since that algorithm is multi-threaded and NodeJS programs cannot be multi-threaded. No. The algorithm is not "multi-threaded". It does not require multiple activities per process.

8. If a process fails in the mutual exclusion algorithm based on quorums, then the following statement IS TRUE:

X	Some processes may block, since some votes in their quorums cannot be collected now. True. Each process should collect a concret set of votes. The participants in each quorum are defined by the algorithm. The goal of the algorithm is to minimise the set of votes to be collected. Then, if any of its participants fails, other processes may block if the failed participant has not yet sent its vote and that missing vote is in their requested quorum.
	All remaining processes go on, since the algorithm is fault tolerant. False. Every process should continue and their continuation relies on the same conditions stated for executions without failures.
	The remaining participants switch immediately to the mutual exclusion algorithm based on a central server, as described in the quorum algorithm. False. The quorum algorithm does not describe any particular behaviour or transition to another algorithm in case of failure of one or more of its participants.
	The hypothesis in this question does not make sense, since processes never fail in that algorithm. False. Unfortunatelly, processes always may fail. The algorithm is not assuming that processes never fail, but that they may fail and recover.

9. Let us assume that we have sent a multi-segment message using a REQ ZeroMQ socket. Which of the following alternatives cannot be used for obtaining all message contents in the 'message' event listener at the REP socket of its receiver?

X	The listener has a single parameter that is managed as an array of segments. False. If the listener has a single parameter, that parameter will only receive the first message segment. All other segments will be lost.

	The listener has as many parameters as segments had the message when it was sent. True. In that case, all segments are received, one per parameter.
	The listener has no parameter. It manages the message segments dynamically using the "arguments" default array. True. In that case, each segment is placed in each slot of the "arguments" array.
	The listener has more parameters than segments had the message when it was sent. True. In that case, some parameters will receive an "undefined" value, but the other ones will get all message segments, one per parameter.

10. Let us imagine that we have a ROUTER-DEALER broker. Which of the following sentences is TRUE?

X	Its ROUTER socket is used as its "front-end" socket in order to forward the response message to the appropriate client process. True. That is the intended functionality and role of the ROUTER socket.
	Its ROUTER socket is used as its "back-end" socket in order to choose the appropriate worker at request forwarding time. False. The ROUTER socket is used in the "front-end" socket. It is not used in the "back-end" one. In a ROUTER-DEALER broker, workers are used following a static round-robin policy, since DEALER sockets use that policy when they are connected with multiple partners.
	If we replace its DEALER socket with a REQ socket, then the broker behaviour is still the same. False. A DEALER socket has an asynchronous behaviour, this means that with a DEALER socket the broker is able to manage multiple workers in a concurrent way; i.e., they may forward multiple requests to workers before the first reply for any of them is received. Using a REQ socket that level of concurrency is impossible. Once the first client request is propagated to a worker, no new request can be forwarded until its reply is received. All additional sending attempts are kept in an internal ZeroMQ buffer. Once that pending reply is received, the first stored request is taken from that buffer and sent to a worker. This compels a sequential and synchronous behaviour, avoiding any concurrency.
	If we replace its ROUTER socket with a REP socket, then the broker behaviour is still the same. False. A ROUTER socket has an asynchronous behaviour, this means that with a ROUTER socket the broker is able to manage multiple requests in a concurrent way; i.e., they may forward multiple requests before the first reply for any of them is received and propagated back to its client. Using a REP socket that level of concurrency is impossible. Once the first client request is received, it is forwarded to a worker. If any additional requests arrive to the REP socket, they are kept in a ZeroMQ internal buffer and they are not delivered to the broker. Only when the reply is received by the broker from a worker, then the broker will propagate it to its client using the REP socket. Once this is done, the first pending request is taken from that buffer and delivered to the broker. This compels a sequential and synchronous behaviour, avoiding any concurrency.