TSR / NIST – Lab 1 (Retake)

This exam consists of 5 multiple choice questions. In every case only one answer is correct. You should answer in a separate sheet. If correctly answered, they contribute 2 points to the exam grade. If incorrectly answered, the contribution is negative: -0.667. So, think carefully your answers.

1. What IS FALSE about the fs.readFile() and fs.readFileSync() operations?

X	By default, operation fs.readFileSync() provides the file contents as a Buffer object, while fs.readFile() provides them in UTF8 coding. False. Both variants report the file contents as a Buffer object by default. On the other hand, in both cases, the programmer may use an optional argument for specifying the coding to be assumed. If any coding is stated, then the file contents are returned as a string with that coding.
	They need a different sequence of arguments. True. The synchronous operations do not have any callback. Since the callback is the last parameter in the asynchronous operations, those sequences of parameters in the operation signatures are different. As a minimum, they differ in the length of that parameter list (in the common case, one unit greater in the asynchronous variant).
	Operation fs.readFileSync() aborts the process in case of error, while fs.readFile() does not abort it. True. When an error happens in a synchronous version of that operation, an exception is raised and the process is aborted. Note that the intended return value of the fs.readFileSync() operation is the content of the file to be read. Because of this, there is no way for communicating the errors in that operation (e.g., a numerical value cannot be used for reporting an error). On the other hand, when the error occurs in the asynchronous variant, its callback is invoked passing an error description in one of its arguments.
	Operation fs.readFile() reports its errors using a callback, while fs.readFileSync() cannot do this. True. The synchronous versions of I/O-based operations do not have any callback parameter. Callbacks are used for both reporting errors and, when no error occurs, to report the result of the operation.

Let us assume the following program in all subsequent questions:

```
var net = require('net');
var _ports = [ 8001,8002,8003,8004,8005];
var _remoteIP =
['158.42.184.5','158.42.4.23','89.238.68.168','158.42.179.56','147.156.222.65'];
var _remotePorts = [80,80,80,8080,80];
var Servers = [];
for (var i = 0; i<_ports.length; i++) {
   Servers.push(net.createServer(function(socket) {
      socket.on('data',function (msg){
          var content;
          try {
               content = JSON.parse(msg);
               if (content.op=='set') {
                 _remoteIP[content.inPort - 8001] = content.remote.ip;
                 _remotePorts[content.inPort - 8001] = content.remote.port;
               }
          } catch(e) {
              // It is not a well-formed JSON string; i.e., it is a regular
              // message to be forwarded.
              var serviceSocket = new net.Socket();
              var pos = socket.localPort - 8001;
```

TSR / NIST – Lab 1 (Retake)

```
            serviceSocket.connect(_remotePorts[pos],_remoteIP[pos], function(){
                  serviceSocket.write(msg);
                  serviceSocket.on('data',function(data){
                    socket.write(data);
                  });
            });
         }
      });
   }));
   Servers[i].listen(_ports[i]);
}
```

2. This program implements...

X	A proxy that manages five ports to which a JSON request may be sent. True. This program implements a proxy that listens to five different ports (8001 to 8005), stored in the "_ports" array. If any message is received from any of those ports, then a new socket is created (called serviceSocket). That socket is connected to a pair remoteIP+remotePort taken from two arrays. The chosen pair is located at the same array index than the port from which the message has been received. The subsequent answer from that server is sent back to the client with the "socket.write(data)" instruction. In some cases, the incoming request is a JSON object to be used for configuring the remote addresses and ports to be used by the proxy. When that request is not a JSON object an exception is raised in the JSON.parse() call. In that case, the behaviour to be followed is the one described in the previous paragraph. Indeed, that is the default behaviour. An example of configuring JSON object is the following: {'op': 'set', 'inPort': 8003, 'remote': {'ip': '213.164.164.171', 'port': '443'} }
	A client that sends five JSON requests to five different servers. False. This program is not a client, but a proxy.
	An array of servers that manage JSON requests. False. This program is not a server, but a proxy.
	A proxy controller that may configure a proxy in five different ports. False. This is not a proxy controller, but a proxy that admits dynamic reconfigurations.

TSR / NIST – Lab 1 (Retake)

3. This program...

X	Listens to requests at ports whose numbers are in an array. True. There are five listened ports, and their numbers are in the "_ports" array.
	Listens to requests at port 8001. This is not a complete answer. Requests can be received in ports 8001 to 8005.
	Listens to requests at port 80. No. Port 80 is not used by this program.
	Sends requests through port 8001. No. It doesn't send any request through that port. Instead, it receives messages from that port and four other ports.

4. In order to change the configuration of the program…

X	A JSON object must be sent with an attribute {'op':'set'}. True. This has been already explained in the justification of question 2.
	A string must be sent, preceding the HTTP request. False. The configuration of the proxy can be changed when the entire contents of an incoming message may be parsed and interpreted as a JSON object. Besides, that object should contain a property called "op" whose value must be "set".
	No configuration change is possible. False. See explanation in statement question 2.
	We must use some command-line arguments when the program is started. False. This program does not process any command-line argument.

5. In this program...

X	When the server replies, that reply is always sent back to the client. True. Replies are processed in the serviceSocket 'data' event listener. That listener has a single instruction: socket.write(data). In that context, the "socket" variable is the client connection object. A write() action on that object sends a reply through that connection; i.e., that message is sent to the client.
	When the server replies, that reply is sent back to the client, unless that reply is a JSON object with a concrete format. False. Replies are not scanned in any way looking for their format. They are always directly sent back to the client.
	A client that interacts with this process must send always a JSON object that precedes the request to be sent to a server. No. Clients do not need to send any JSON object. Besides, if a JSON object is sent by any process to the proxy, that object should be the unique content in that "configurator" message.
	HTTP requests (send by browsers) are not properly managed, since they are not JSON objects. No. Indeed, they are handled without any trouble. Their management is implemented in the "catch" block that fills the second half of this program.