TSR – 16 octubre 2017. EXERCICI 1

Escriga en NodeJS dos programes (client.js i servidor.js) que complisquen aquests requisits:

1) Llur comunicació ha de basar-se en *sockets,* utilitzant el mòdul "net".
2) El programa client rebrà dos arguments des de la línia d'ordres. El primer indica quants missatges s'enviaran al servidor i el segon és una cadena a enviar en cada missatge. Es concatenarà el nombre de missatge actual a aquesta cadena.
3) El servidor retorna una resposta (la cadena 'Ok') per a cada missatge rebut. Ha de mostrar el missatge rebut en pantalla.
4) El client també mostrarà cada resposta rebuda en pantalla.
5) Una vegada el client haja enviat tots els seus missatges i rebut la darrera resposta, tancarà la connexió i finalitzarà la seua execució.
6) El client enviarà un missatge cada segon.
7) El servidor ha d'acceptar connexions des de múltiples clients i ha d'utilitzar el port 9000.
8) Els clients i el servidor s'executaran al mateix ordinador.

Exemple d'execució:

```
$ node servidor &
$ node client 3 "Un missatge " &
$ node client 2 "Altre client " &
$ Un missatge 1
Ok
Altre client 1
Ok
Un missatge 2
Ok
Altre client 2
Ok
Un missatge 3
Ok
```

Referencia bàsica:

- The `process.argv` property returns an array containing the command line arguments passed when the Node.js process was launched.
- net.createServer([options][, connectionListener]). Creates a new server. The `connectionListener` argument is automatically set as a listener for the `'connection'` event. It returns a net.Server socket object.
- net.connect(port[, host][, connectListener]). A factory function, which returns a new net.Socket and automatically connects to the supplied port and host. If host is omitted, 'localhost' will be assumed. The connectListener parameter will be added as a listener for the 'connect' event once.
- net.Socket.write(data[, encoding][, callback]). Sends data on the socket. The second parameter specifies the encoding in the case of a string--it defaults to UTF8 encoding. Returns true if the entire data was flushed successfully to the kernel buffer. Returns false if all or part of the data was queued in user memory. 'drain' will be emitted when the buffer is again free. The optional callback parameter will be executed when the data is finally written out - this may not be immediately.
- net.Socket.end([data][, encoding]). Half-closes the socket. i.e., it sends a FIN packet. It is possible the server will still send some data. If data is specified, it is equivalent to calling socket.write(data, encoding) followed by socket.end().
- net.Server.listen([port][, hostname][, backlog][, callback]). Begin accepting connections on the specified port and hostname. If the hostname is omitted, the server will accept connections on any IP address. Backlog is the maximum length of the queue of pending connections. The default value of this parameter is 511 (not 512). This function is asynchronous. When the server has been bound, 'listening' event will be emitted. The last parameter callback will be added as a listener for the 'listening' event. One issue some users run into is getting EADDRINUSE errors. This means that another server is already running on the requested port.

SOLUCIÓ

```
// Client.js
const net=require('net');
var socket=net.connect(9000);
var args=process.argv.slice(2);

if (args.length!=2) {
        console.log("Usage: node %s <num-messages> <msg-text>",process.argv[1])
        process.exit(1);
}

const numMsgs=parseInt(args[0]);
const msgText=args[1];
var receivedReplies=0;
var msgCounter=0;

socket.on('data', function(msg) {
        console.log(msg+'');
        receivedReplies++;
        if (receivedReplies==numMsgs) {
                socket.end();
                process.exit(0);
        }

});

setInterval( function () {
        msgCounter++;
        socket.write(msgText+" "+msgCounter);
},1000);
```

```
// Server.js
const net=require('net');
var socket=net.createServer(function (con) {
        con.on('data', function (msg) {
                console.log(msg+'');
                con.write('Ok');
        });
});

socket.listen(9000);
```

TSR – 16 octubre 2017. EXERCICI 2

Es necessita un programa que reba des de la línia d'ordres una llista de noms de fitxer. La seua eixida ha de ser el nom i grandària del major d'aquests fitxers.

Dos programadors han escrit les següents propostes de solució per a aquest problema:

```
// Program1.js
const fs=require('fs');
var args=process.argv.slice(2);
var maxName='NONE';
var maxLength=0;
for (var i=0; i<args.length; i++)
        fs.readFile(args[i],'utf8', function(err,data) {
                if (!err) {
                        console.log('Processing %s...',args[i]);
                        if (data.length>maxLength) {
                                maxLength=data.length;
                                maxName=args[i];
                        }
                }
        });
console.log('The longest file is %s and its length is %d bytes.', maxName, maxLength);
```

```
// Program2.js
const fs=require('fs');
var args=process.argv.slice(2);
var maxName='NONE';
var maxLength=0;
function generator(name,pos) {
   return function(err,data) {
        if (!err) {
           console.log('Processing %s...',name);
           if (data.length>maxLength) {
                maxLength=data.length;
                maxName=name;
           }
        }
        if (pos==args.length-1)
           console.log('The longest file is %s and its length is %d bytes.', maxName, maxLength);
   }
}
for (var i=0; i<args.length; i++)
        fs.readFile(args[i],'utf8', generator(args[i],i));
```

Supose que ambdòs programes han sigut utilitzats en un directori amb aquest contingut:

```
$ ls -go
total 63368
-rw-rw-r-- 1 64880640 Oct  4 15:57 b
-rw-rw-r-- 1      495 Oct  4 16:09 Program1.js
-rw-rw-r-- 1      660 Oct  4 16:10 Program2.js
```

...mitjançant aquestes ordres:

a) node Program1 b Program1.js Program2.js

b) node Program2 b Program1.js Program2.js

Justifique quina eixida mostraran (quins missatges i en quin ordre) les dues execucions. Ambdòs programes són sintàcticament correctes i no generen cap excepció o error sintàctic mentre s'executen.

SOLUCIÓ

a)

Els missatges mostrats en pantalla per aquest primer procés són:

The longest file is NONE and its length is 0 bytes.
Processing undefined...
Processing undefined...
Processing undefined...

Analitzem per què es mostren en aquest ordre:

- Cada fitxer es llig de manera asincrònica. Això implica que les lectures són gestionades internament per altres fils d'execució. En algun moment cada fil finalitzarà la seua lectura i mostrarà el seu resultat, executant el *callback* corresponent. Aquest *callback* presenta un missatge "Processing <nom-fitxer>..."
- D'altra banda, l'última línia del programa s'executa de manera sincrònica en el fil principal. Aleshores, el seu missatge es veu el primer. En ell, maxName i maxLength encara tenen els seus valors inicials. Això explica el contingut del primer missatge: "The longest file is NONE and its length is 0 bytes".
- Posteriorment, cada *callback* és executat. Quan això ocorre, el bucle "for" utilitzat per a cridar fs.readFile() haurà completat totes les seues iteracions. Aleshores, la variable "i" té un valor 3. Això explica que la instrucció "console.log()" usada als *callbacks* sempre mostre el mateix nom de fitxer: el que estiga en "args[3]". El vector "args" només guarda tres noms en aquest exemple, dins de les components 0 a 2, inclusivament. Així, "args[3]" no manté cap valor i el seu contingut és "undefined". Aquesta és la informació mostrada en els darrers tres missatges.

b)

Els missatges que podrien mostrar-se en aquest segon procés serien:

Processing Program1.js...
Processing Program2.js...
The longest file is Program2.js and its length is 660 bytes.
Processing b...

Analitzem el seu contingut i ordre:

- En aquest programa les instruccions "console.log()" que mostren els missatges estan totes dins del *callback* de fs.readFile(). A més, com aquest *callback* necessita només dos paràmetres (l'error i el contingut del fitxer, respectivament) i nosaltres volem utilitzar un nom de fitxer (per a mostrar-lo) i un índex (per a comparar-lo i sabes si s'ha de mostrar el missatge final), s'ha usat una altra funció per a generar el *callback* i proporcionar la clausura que mantinga aquestes dues dades.
- Amb generator() assegurem que els missatges "Processing <nom-fitxer>..." mostren un nom de fitxer correcte. Això implica que en l'eixida vista dalt, els *callbacks* van ser executats en aquest ordre: (1) Program1.js, (2) Program2.js, (3) b. El seu ordre concret depén de diversos factors: la grandària dels fitxers i la política de planificació del sistema operatiu són els dos principals. La gran longitud del fitxer "b" (almenys si es compara amb la dels altres fitxers) implica que es mostre al final. D'altra banda, les posicions de Program1.js i Program2.js depenen de la política de planificació per a les peticions fetes al sistema de fitxers. Així, l'ordre d'aquests dos missatges té cert nivell d'incertesa, encara que l'ordre més probable és el que s'ha vist dalt.
- El missatge "The longest file..." es mostra quan el *callback* correspon al fitxer "Program2.js" (just després d'haver escrit "Processing Program2.js...") ja que això ocorre quan el valor de l'iterador és "args.length-1" (és a dir, 2). En la nostra traça no escriurà el resultat correcte (és a dir, que el major fitxer és "b" i la seua longitud és 64880640 octets) perquè el *callback* associat a "b" encara no ha arribat a executar-se.