TSR - PRÀCTICA 1
(Segona part)

NODEJS

La pràctica 1 consisteix en 2 parts, una part dedicada a JavaScript d'1 sessió i una part dedicada a NodeJS de 2 sessions. **Aquest document descriu la segona part de la pràctica, dedicada a NodeJS**.

1. Introduir els procediments i les eines necessàries per al treball en el laboratori de TSR.
2. Introduir tècniques bàsiques per a la programació i desenvolupament en JavaScript i NodeJS.

INTRODUCCIÓ

Eines

Les pràctiques es desenvolupen a les **màquines virtuals del portal**, en JavaScript sobre l'entorn NodeJS. Això elimina possibles conflictes derivats de la compartició de recursos en poliLabs.

Qualsevol alumne pot instal·lar un entorn similar en els seus propis equips en Windows, LINUX o MacOS. Consultar http://nodejs.org

Procediments

En general escrivim amb qualsevol editor el codi JavaScript en un fitxer amb l'extensió js (x.js), i executem aquest codi amb l'ordre **node x[.js] [argsOpcionales]**

Per a minimitzar els possibles errors, es recomana:

- utilitzar el mode estricte: `node --use_strict fitxer.js`
- documentar correctament cada funció d'interfície
    - significat i tipus de cada argument, així com qualsevol restricció addicional
    - quin tipus d'errors operacionals poden aparèixer, i com es gestionaran
    - el valor de retorn

PRÀCTICA 1 (SEGONA PART): NODEJS

Per a aquesta segona part disposem de 2 sessions. Es proposa un treball a realitzar per a cadascuna de les sessions. No obstant això l'alumne pot gestionar-se el seu temps i dedicar més o menys a cadascuna de les sessions que proposem per a acabar de completar tots els aspectes que s'esmenten.

- **Sessió 1.2: Mòduls de NodeJS.**
- **Sessió 1.3: Proxy invers.**

SESSIÓ 1.2: MÒDULS DE NODEJS.

NodeJS inclou gran varietat de mòduls que proporcionen funcionalitat útil per a treballar mitjançant fitxers, amb la xarxa, amb aspectes de seguretat, etc. En aquesta sessió treballarem amb alguns dels mòduls més rellevants.

Es tractaran els mòduls: **fs, events, http, net.**

Durant aquesta sessió hauràs de practicar amb el codi que es proporciona com a exemple per a cadascun d'aquests mòduls, i contestar les qüestions o realitzar les activitats que es proposen.

1.2.1 MÒDUL FS: ACCÉS A FITXERS

Tots els mètodes corresponents a operacions sobre fitxers apareixen en el mòdul fs.js. Les operacions són asincròniques, però per a cada funció asincrònica **xx** sol existir la variant sincrònica **xxSync**. Els següents codis poden ser copiats i executats.

Llig els següents apartats A, B i C d'aquesta secció. Per a cadascun d'ells copia el codi, analitza'l i executa'l fins que ho comprengues. Al final hi ha una sèrie de preguntes que has de contestar.

A. Llegir asincrònicament el contingut d'un fitxer (**`read1.js`**)

```
const fs = require('fs');
fs.readFile('/etc/hosts', 'utf8', function (err,data) {
  if (err) {
    return console.log(err);
  }
  console.log(data);
});
```

B. Escriure asincrònicament el contingut en un fitxer (**`write1.js`**)

```
const fs = require('fs');
fs.writeFile('/tmp/f', 'contingut del nou fitxer', 'utf8',
  function (err) {
  if (err) {
    return console.log(err);
  }
  console.log("s'ha completat l'escriptura");
});
```

C. Escriptures i lectures adaptades. Utilització de mòduls.

Definim el mòdul **`fiSys.js`**, basat en **`fs.js`**, per a accedir a fitxers juntament amb alguns exemples del seu ús.

(**`fiSys.js`**)

```
//Mòdul fiSys
//Exemple de mòdul de funcions adaptades per a l'ús de fitxers.
//(Podrien haver-se definit més funcions.)

const fs=require("fs");

function readFile(fitxer,callbackError,callbackLectura){
        fs.readFile(fitxer,"utf8",function(error,dades){
                if(error) callbackError(fitxer);
                        else callbackLectura(dades);
        });
}


function readFileSync(fitxer){
        var resultat; //retornarà undefined si ocorre algun error en la lectura
        try{
                resultat=fs.readFileSync(fitxer,"utf8");
        }catch(e){};
        return resultat;
}


function writeFile(fitxer,dades,callbackError,callbackEscriptura){
        fs.writeFile(fitxer,dades,function(error){
                if(error) callbackError(fitxer);
                        else callbackEscriptura(fitxer);
        });
}

exports.readFile=readFile;
exports.readFileSync=readFileSync;
exports.writeFile=writeFile;
```

Programa de lectura mitjançant el mòdul "fiSys" (**`read2.js`**)

```
//Lectures de fitxers

const fiSys=require("./fiSys");


//Per a la lectura asincrònica:
console.log("Invocació lectura asincrònica");
fiSys.readFile("/proc/loadavg",cbError,format);
console.log("Lectura asincrònica invocada\n\n");

//Lectura sincrònica
console.log("Invocació lectura sincrònica");
const dades=fiSys.readFileSync("/proc/loadavg");
if(dades!=undefined)format(dades);
        else console.log(dades);
console.log("Lectura sincrònica finalitzada\n\n");

//- - - - - - - - - - -
function format(dades){
        const separador=" "; //espai
        const tokens = dades.toString().split(separador);
        const min1 = parseFloat(tokens[0])+0.01;
        const min5 = parseFloat(tokens[1])+0.01;
        const min15 = parseFloat(tokens[2])+0.01;
        const resultat=min1*10+min5*2+min15;
        console.log(resultat);
}

function cbError(fitxer){
        console.log("ERROR DE LECTURA en "+fitxer);
}
```

Programa d'escriptura mitjançant el mòdul fiSys (**`write2.js`**)

```
//Escriptura asincrònica de fitxers

const fiSys = require('./fiSys');

fiSys.writeFile('text.txt','contingut del nou fitxer',cbError,cbEscriptura);

function cbEscriptura(fitxer){
        console.log("Escriptura realitzada en: "+fitxer);
}

function cbError(fitxer){
        console.log("ERROR D'ESCRIPTURA en "+fitxer);
}
```

Qüestions

1. Raona quantes iteracions del bucle d'esdeveniments (torns) succeeixen en cadascun dels programes analitzats.

2. En l'apartat C s'ha desenvolupat un mòdul d'usuari. Detalla els passos que s'han realitzat en el codi per a crear un mòdul, a diferència del codi necessari per a programar una aplicació.

3. Realitza 2 versions noves de l'apartat A. Una mitjançant promeses i una altra mitjançant async/await. Utilitza el material de teoria com a base.

1.2.2 MÒDUL EVENTS

El mòdul "events" proporciona una certa funcionalitat per a l'ús d'esdeveniments en NodeJS. És destacable el fet que diversos mòduls de NodeJS utilitzen aquest mòdul per a gestionar esdeveniments. Se't proporcionen dos programes que has de comprendre i has de completar el tercer amb el teu propi codi.

A. Programa que usa el mòdul "events". (**`emisor1.js`**)

Analitza i executa el programa, fins a comprendre el seu funcionament.

```
const ev = require('events')              // library import (Using events module)

const emitter = new ev.EventEmitter()     // Create new event emitter
const e1='print', e2='read'                // identity of two different events

function handler (event,n) {    // function declaration, dynamic type args, higher-order
function
    return () => { // anonymous func, parameterless listener, closure
      console.log(event + ':' + ++n + ' times')
    }
}

emitter.on(e1, handler(e1,0)) // listener, higher-order func (callback)
emitter.on(e2, handler(e2,0)) // listener, higher-order func (callback)
emitter.on(e1, ()=>{console.log('something has been printed')}) //several listeners on
e1

emitter.emit(e1) // emit event
emitter.emit(e2) // emit event

console.log('---------------------------')
setInterval(()=>{emitter.emit(e1)}, 2000) // asynchronous (event loop), setInterval
setInterval(()=>{emitter.emit(e2)}, 8000) // asynchronous (event loop), setInterval
console.log('\n\t========> end of code')
```

B. Programa que usa el mòdul "events". (**emisor2.js**)

Analitza aquest altre exemple (emisor2.js). En generar esdeveniments es poden generar valors associats (arguments per al 'oïdor' de l'esdeveniment).

```
const ev = require('events')

const emitter = new ev.EventEmitter()
const e1='e1', e2='e2'

function handler (event,n) {
   return (incr)=>{ // listener with param
      n+=incr
      console.log(event + ':' + n)
   }
}

emitter.on(e1, handler(e1,0))
emitter.on(e2, handler(e2,'')) // implicit type casting

console.log('\n\n---------------------------- init\n\n')
for (let i=1; i<4; i++) emitter.emit(e1,i) // sequence, iteration, generation with param
console.log('\n\n---------------------------- intermediate\n\n')
for (let i=1; i<4; i++) emitter.emit(e2,i) // sequence, iteration, generation with param
console.log('\n\n---------------------------- end')
```

Activitat

Se't proporciona el codi del programa emisor3, i l'has de completar perquè complisca unes certes especificacions que detallarem a continuació.

Codi a completar (**emisor3.js**)

```
...
const e1='e1', e2='e2'
let inc=0, t

function rand() { // ha de retornar valors aleat en rang [2000,5000) (ms)
   ... // Math.floor(x) retorna la part sencera del valor x
   ... // Math.random() retorna un valor en el rang [0,1)
}

function handler (e,n) { // e és l'esdeveniment, n el valor associat
   return (inc) => {..} // l'oïdor rep un valor (inc)
}

emitter.on(e1, handler(e1,0))
emitter.on(e2, handler(e2,''))

function etapa() {
   ...
}

setTimeout(etapa,t=rand())
```

Completa el codi (afegint codi en lloc dels punts suspensius) sense eliminar codi perquè complisca amb el següent:

El codi ha de cridar una primera vegada a la funció "etapa". Dins de la funció "etapa" s'ha de reprogramar la crida a "etapa" perquè el programa execute una sèrie de "etapes".

Cada etapa ha d'iniciar-se amb un retard d'entre 2 i 5 segons. Cada etapa ha de fer el següent:

- Emetre els esdeveniments e1 i e2, passant com a valor associat el de la variable "inc".
- Augmentar la variable "inc" una unitat
- Mostrar per consola el retard en l'execució de l'etapa.

Exemple d'execució (únicament el fragment inicial)

```
e1 --> 0
e2 --> 0
etapa 1 iniciada després de 3043 ms
e1 --> 1
e2 --> 01
etapa 2 iniciada després de 3869 ms
e1 --> 3
e2 --> 012
etapa 3 iniciada després de 2072 ms
e1 --> 6
e2 --> 0123
etapa 4 iniciada després de  2025 ms
e1 --> 10
e2 --> 01234
etapa 5 iniciada després de 2325 ms
```

Qüestions

El codi que s'ha proposat per a "emisor3", ha de reprogramar cada nova etapa dins de la pròpia etapa. Raona quines implicacions tindria programar totes les etapes de colp amb un codi similar al següent:

```
t = 0

for (let i=0; i<=10; i++) {

      t = t + rand();

      setTimeout (etapa, t);

}
```

1.2.3 MÒDUL HTTP

Mòdul amb funcions per a desenvolupament de servidors Web (servidors HTTP)

Se't demana que analitzes i comproves el funcionament del següent programa que empra el mòdul "http".

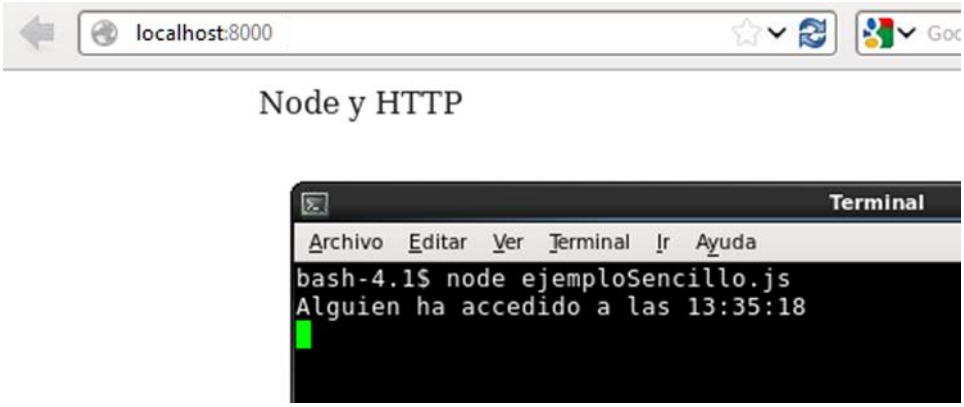
Servidor web (**`ejemploSencillo.js`**) que saluda al client

Codi	comentari
`const http = require('http');`	Importa mòdul http
`function dd(i) {return (i<10?"0":"")+i;}`	dd(8) -> "08" dd(16) -> "16"

`const server = http.createServer(`
`  function (req,res) {`
`  res.writeHead(200,{'Content-Type':'text/html'});`
`  res.end('<marquee>Node i Http</marquee>');`
`  var d = new Date();`
`  console.log('Algú ha accedit a les '+`
`  d.getHours() +":"+`
`  dd(d.getMinutes()) +":"+`
`  dd(d.getSeconds()));`
`}).listen(8000);`
	crea el servidor i
	li associa aquesta funció
	que retorna una resposta fixa
	i a més escriu l'hora en la consola
	
	El servidor escolta en el port 8000

Executa el servidor i utilitza un navegador web com a client

- Accedeix a la URL **`http://localhost:8000`**
- Comprova en el navegador la resposta del servidor, i en la consola el missatge escrit pel servidor



1.2.4 MÒDUL NET

Aquest mòdul conté l'API per a emprar sockets bàsics TCP/IP.

Es proporcionen dos programes, client i servidor. Analitza el codi de tots dos i executa'ls diverses vegades per a observar el seu funcionament.

Client (`netClient.js`)

```
const net = require('net');

const client = net.connect({port:8000},
    function() { //connect listener
        console.log('client connected');
        client.write('world!\r\n');
    });

client.on('data',
    function(data) {
        console.log(data.toString());
        client.end(); //no more data written to the
stream
    });

client.on('end',
    function() {
        console.log('client disconnected');
    });
```

Servidor (`netServer.js`)

```
const net = require('net');

const server = net.createServer(
    function(c) { //connection listener
        console.log('server: client connected');
        c.on('end',
            function() {
                console.log('server: client disconnected');
            });
        c.on('data',
            function(data) {
                c.write('Hello\r\n'+ data.toString()); // send
resp
                c.end(); // close socket
            });
    });

server.listen(8000,
    function() { //listening listener
        console.log('server bound');
    });
```

Activitat

Es demana que modifiques el codi d'aquests programes client i servidor per a aconseguir un servei que proporcione la càrrega de l'ordinador servidor. Utilitzant "netClient" com a base, crea un programa "netClientLoad". Utilitzant el programa "netServer", crea un programa al qual anomenarem "netServerLoad".

Per a fer aquesta activitat has de realitzar 2 parts:

Part 1: modificar el programa "netClientLoad" que acabes de crear, perquè utilitze correctament la línia d'arguments.

Part 2: Per a actualitzar el programa "netServerLoad", se't proporciona la funció "getLoad()" que pots integrar en el seu codi

Part 1 de l'activitat – Accés a arguments en línia d'ordres

El shell arreplega tots els arguments en línia d'ordres i li'ls passa a l'aplicació JS empaquetats en un array denominat process.argv (abreviatura de 'argument values'), per la qual cosa podem calcular la seua longitud i accedir a cada argument per la seua posició

- **`process.argv.length:`** nombre d'arguments passats per la línia d'ordres
- **`process.argv[i]:`** permet obtenir l'argument i-èssim. Si hem usat l'ordre "**`node programa arg1 …`**" llavors **`process.argv[0]`** conté la cadena '**`node`**', **`process.argv[1]`** conté la cadena '**`programa`**', **`process.argv[2]`** la cadena '**`arg1`**', etc.

Pot utilitzar-se **args=process.argv.slice(2)** per a descartar 'node' i el path del programa, de manera que en **args** només quedaran els arguments reals per a l'aplicació encapsulats i accessibles com a elements d'un array, a partir de la posició 0.

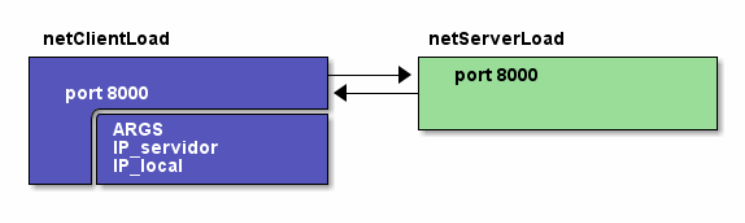
Part 2 de l'activitat. Consulta la càrrega de l'equip

Pots integrar la funció "getLoad()" en el codi de netServerLoad, perquè obtinga la seua càrrega actual cada vegada que un client demane el servei. Note's que només funciona en Linux:

```
function getLoad(){
  data=fs.readFileSync("/proc/loadavg"); //requereix fs
  var tokens = data.toString().split(' ');
  var min1  = parseFloat(tokens[0])+0.01;
  var min5  = parseFloat(tokens[1])+0.01;
  var min15 = parseFloat(tokens[2])+0.01;
  return min1*10+min5*2+min15;
};
```

A títol de curiositat, perquè entenguem una mica el codi d'aquesta funció, podem observar el que fa: Aquesta funció llig dades del fitxer /proc/loadavg. Aquest pseudo-fitxer conté informació relativa a la càrrega del sistema Linux. Per a proporcionar la càrrega actual, la funció filtra els valors que li interessa (els suma una centèsima per a evitar la confusió entre el valor 0 i un error), i els processa calculant una mitjana ponderada (pes 10 a la càrrega de l'últim minut, pes 2 als últims 5 minuts, pes 1 als últims 15)

Descripció general de tots dos programes



- El servidor **`netServerLoad`** no rep arguments en línia d'ordres i escoltarà en un cert port (el port 8000 per exemple).
- El client **`netClientLoad`** ha de rebre com a arguments en línia d'ordres l'adreça IP del servidor i la seua IP local. Connectarà amb el servidor en l'adreça IP del servidor i el port del servidor.
- Protocol: Quan el client envia una petició al servidor, inclou la seua pròpia IP, el servidor calcula la seua càrrega i retorna una resposta al client en la qual inclou la pròpia IP del servidor i el nivell de càrrega calculat amb la funció **getLoad**.
- Pot ser una bona idea que tots dos programes intercanvien les dades mitjançant JSON. Revisa l'API "JSON.stringify()" i "JSON.parse()"
- Cal assegurar-se que el client finalitza en obtenir la càrrega (ex. amb **`process.exit()`**) o tancant la seua connexió.
- Es pot esbrinar la IP des de la interfície web del portal, o usant l'ordre **ip addr**, o **ifconfig**
- Completa tots dos programes, col·loca'ls en equips diferents col·laborant amb algun company (o connectant mitjançant ssh), i fes que es comuniquen mitjançant el port 8000: **`netServerLoad`** ha de calcular la càrrega com a resposta a cada petició rebuda des del client, i **`netClientLoad`** ha de mostrar la resposta en pantalla.

SESSIÓ 1.3: PROXY INVERS.

Un intermediari o proxy és un servidor que en ser invocat pel client redirigeix la petició a un tercer, i posteriorment encamina la resposta final de nou al client.

- Des del punt de vista del client, es tracta d'un servidor normal (oculta al servidor que realment completa el servei)
- Pot residir en una màquina diferent a la del client i la del servidor
- L'intermediari pot modificar ports i direccions, però no altera el cos de la petició ni de la resposta

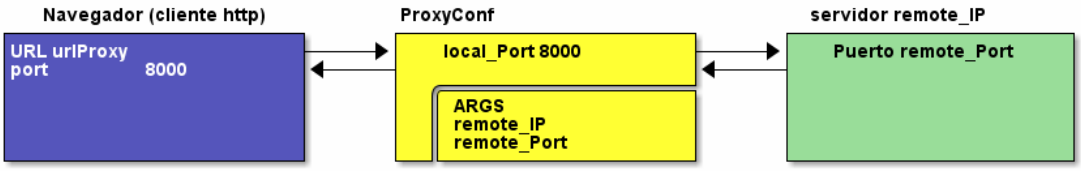
És la funcionalitat que ofereix un redirector, com un proxy HTTP, una passarel·la ssh o un servei similar. Més informació en http://en.wikipedia.org/wiki/proxy_server

En aquesta sessió de pràctiques treballarem amb 3 versions de proxy. Et donem la primera versió ja implementada i has de completar les altres 2. En finalitzar has de respondre a les qüestions que plantegem.

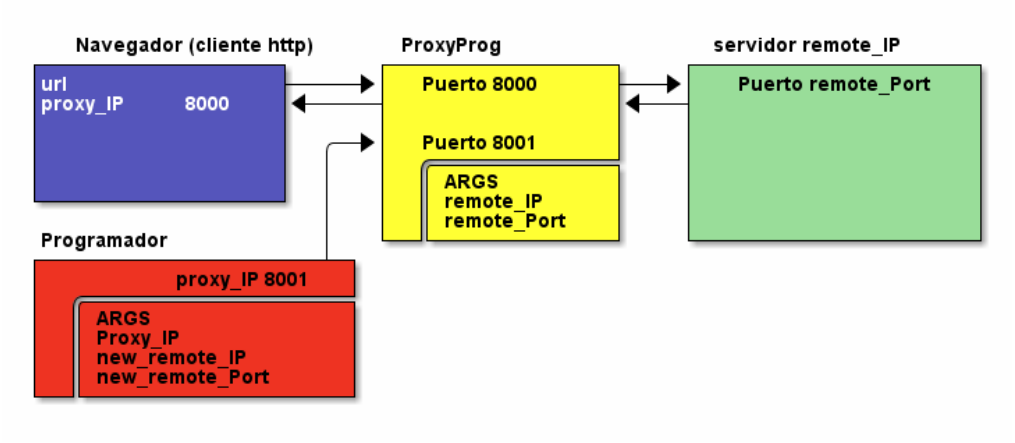
Les tres versions a considerar són les següents:

1. Proxy bàsic (**Proxy**). Quan el client http (navegador) contacta amb el proxy en el port 8000, el proxy redirigeix la petició al servidor web memex (158.42.185.55 port 8080), i posteriorment retorna la resposta del servidor a l'invocador

2. Proxy configurable (**ProxyConf**): en lloc d'IP i port remots fixos en el codi, els rep com a arguments en línia d'ordres



3. Proxy programable (**ProxyProg**). Els valors d'IP i port del servidor es prenen inicialment des de línia d'ordres, però posteriorment es modifiquen en rebre missatges del programador en el port 8001



Et proporcionem el codi de Proxy bàsic: analitza-ho fins a comprendre el seu funcionament

Codi (Proxy.js)	comentaris

`const net = require('net');`
 	
`const LOCAL_PORT = 8000;`
`const LOCAL_IP = '127.0.0.1';`
`const REMOTE_PORT = 80;`
`const 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);`
`console.log("TCP server accepting connection on port: " + LOCAL_PORT);`
	Usa un socket per a dialogar amb el client (socket) i un altre per a dialogar amb el servidor (serviceSocket)
	
	1.- llig un missatge (`msg`) del client
	2.- obri una connexió amb el servidor
	3.- escriu una còpia del missatge
	4.- espera la resposta del servidor i retorna una còpia al client

1.- Proxy bàsic: En aquesta primera part no has de programar res. Has de comprovar el seu funcionament. Comprova el funcionament del proxy usant un navegador web que apunte a http://adreça_del_proxy:8000/

Realitza diferents proves i analitza el codi que se't proporciona.

També pots provar el proxy perquè mitjance a un client netClientLoad i un servidor netServerLoad.

2.- Proxy configurable: Prenent com a base el proxy bàsic, crea el proxy configurable (ProxyConf.js). L'única cosa a realitzar és analitzar la línia d'ordres per a obtenir d'ella la direcció del servidor al qual connectarà el proxy.

Una vegada ho tinguem fet podem fer diverses proves, iguals a les proves que vam fer en la versió anterior. Realitza almenys les 2 que proposem:

- 1. Mitjançar un servidor WEB. Per exemple, mitjançar l'accés al servidor de la UPV (continuem utilitzant com a port local d'atenció de peticions el port 8000)

- 2. Mitjançar l'accés entre netClientLoad i netServerLoad.
  - Si ProxyConf s'executa en la mateixa màquina que netServerLoad i tenim una col·lisió en l'ús de ports -> modifica el codi de netServerLoad perquè use un altre port. També pots modificar el codi de netServerLoad perquè reba com a argument el port del qual ha d'escoltar.
  - Si en executar el programa apareix l'error "EADDRINUSE", indica que estem referenciant un port que ja està en ús per part d'un altre programa

3. Proxy programable:

Utilitzant com a base el proxy configurable, creem el proxy programable (ProxyProg.js)

Hem d'implementar el codi del programador (programador.js) i fer alguns canvis al codi del nou proxy. Per a implementar el programador podem prendre com a base el programa client netClientLoad i fer les modificacions necessàries. Per la seua part per a fer que el nou Proxy programable reba peticions del programador, podem prendre part del codi del servidor que tenim realitzat (netServerLoad)

Hem de completar tots dos programes perquè es contemplen els següents aspectes:

- El programador ha de rebre en línia d'ordres l'adreça IP del proxy, i els nous valors d'IP i port corresponents al servidor. Amb la IP del proxy i el port per defecte del proxy (port 8001), contactarà amb el proxy per a enviar-li les dades del servidor remot.

- El programador codificarà els valors i els remetrà com a missatge al proxy, després de la qual cosa acaba. Per part seua, el proxy rebrà aquest missatge per a actualitzar l'adreça del servidor al qual contactarà a partir d'aqueix moment.

- El **`programador.js`** hauria d'enviar missatges amb un contingut com el següent:

```
var msg = JSON.stringify ({'remote_ip':"158.42.4.23", 'remote_port':80})
```

Pots experimentar usant com a servidors, dos servidors WEB diferents i anar comprovant com el programador modifica el servidor destí. També podries emprar 2 servidors netServerLoad i 1 client netClientLoad, més el programador.

Qüestions.

1. Quins avantatges observes en la interacció entre un client i un servidor en emprar JSON respecte a no emprar-lo?

2. Imagina un nou proxy que disposa d'un pool de servidors als quals enviar peticions. Com podríem aconseguir que aquest nou proxy s'encarregue d'enviar peticions al servidor que estiga menys carregat a cada moment? Raona com desenvoluparies aquest nou proxy.