TSR - PRÀCTICA 1 CURS 2021/22

INTRODUCCIÓ AL LABORATORI: JAVASCRIPT I NODEJS

Aquesta pràctica consta de tres sessions, a fer en sessions d'escriptori LINUX, i té els següents objectius:

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.

És necessari haver tingut un contacte previ amb aquest tipus d'activitats, propòsit cobert per la `Pràctica 0`.

- Entre el material disponible en la carpeta corresponent[1] de l'àrea de Recursos de PoliformaT convé destacar els documents d'introducció als laboratoris del DSIC i d'introducció al laboratori de TSR. Entre els dos desciruen l'operativa per a les pràctiques de l'assignatura.
- Cal destacar que en la redacció d'aquest butlletí se suposa que usarem les sessions d'escriptori al laboratori o mitjançant **Polilabs**, però és senzill reinterpretar-ho per a la seua aplicació en altres escenaris...
    - La imatge equivalent per a VirtualBox
    - El vostre servidor virtual de portal
    - Directament sobre el vostre ordinador personal (si disposes de tots els requisits).

[1] Consulta l'apartat final: "**Referències**"

INTRODUCCIÓ

Conflictes i eines

Les pràctiques es desenvolupen en JavaScript sobre l'entorn NodeJS

Quan treballem amb els equips d'escriptori dels laboratoris del DSIC, realment ho fem en sessions que s'inicien en màquines virtuals compartides. Aquesta característica **no és compatible** amb l'**ús de ports** o altres recursos exclusius, particularment en NodeJS, raó per la qual es necessitarà alguna organització que impedisca la interferència entre el treball de diversos alumnes.

- En resum: els usuaris i usuàries competeixen per l'ús dels mateixos ports quan es troben resolent el mateix problema.

Els números de port que apareixen en els enunciats seran reinterpretats i modificats perquè no coincidisquen entre alumnes diferents. Atès que en el laboratori hem reservat el rang 50000 a 59999 (el 5 és fix), i que cap exercici requereix usar més de 10 ports simultàniament (els representaríem amb la xifra de la unitat, de 0 a 9), ens queden 3 xifres per a diferenciar cada alumne. Un criteri consisteix a prendre els últims 3 dígits del DNI.

- Així, una alumna amb DNI 29332481 disposarà dels ports 54810 a 54819, amb *la quasi* seguretat de no entrar en conflicte amb altres companys i companyes.
- En un exercici que cite els ports 8000, 8001 i 8002, l'alumna anterior pot reinterpretar els ports com 54810 (en comptes de 8000), 54811 (en comptes de 8001) i 54812 (en lloc de 8002).

Les característiques d'aquestes màquines d'escriptori compartides són:

- LINUX (Ubuntu de 64 bits), accés al vostre emmagatzematge en xarxa.
- Entorn NodeJS (versió 12).
- Biblioteques auxiliars bàsiques (fs, http, net, ...)
- Gestor de paquets npm (permet instal·lar qualsevol altra biblioteca de NodeJS)
  - Necessitaràs executar “npm init” si ho instal·les pel teu compte
- Editors de textos (en el menú Programació). Podem utilitzar qualsevol que incloga ressaltat sintàctic, però en la Pràctica 0 ja es va recomanar Visual Studio Code.

Qualsevol alumne pot instal·lar un entorn similar en els seus propis equips en Windows, LINUX o MacOS, sense les limitacions descrites quant a ports. Consultar http://nodejs.org

Procediments

Clàusula de bon ús del laboratori

Els recursos que es posen a la disposició de l'alumne es justifiquen pel seu caràcter educatiu. La correcta utilització dels mateixos és responsabilitat de l'alumne, que es compromet a:

- Vetlar per la confidencialitat de les seues claus d'accés.
- No interferir en l'activitat dels altres companys.
- No utilitzar aquests recursos per a finalitats diferents a les descrites en els butlletins de pràctiques

En les instal·lacions existeixen mecanismes que monitoren l'ús dels recursos i aporten informacions que poden ser consultades amb posterioritat.

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] [argsOpcionals]**

Aquesta pràctica consta de tres sessions que s'organitzen en dues parts: la primera, dedicada a JavaScript; i la segona, a NodeJS. El seu desenvolupament serà guiat pel professor.

Ha de prendre's en compte que durant el desenvolupament de les pràctiques poden aparèixer diferents **tipus d'errors**. Alguns errors freqüents són els següents.

Error Sintàctic		
Definició	Detecció/solució	Exemple
Sintaxi il·legal (identificador no vàlid, estructures mal niades, etc)	En interpretar el codi s'indica tipus d'error i la seua ubicació en el codi. Corregir sobre codi font	> "creïlla"(); TypeError: string is not a function at repl:1:9 at REPLServer.defaultEval (repl.js:132:27) at bound (domain.js:254:14) at REPLServer.runBound [as eval] (domain.js:267:12) at REPLServer.<anonymous> (repl.js:279:12) ....
Error Lògic		
Definició	Detecció/solució	Exemple
Error de programació (intentar accedir a una propietat inexistent o 'undefined', passar com a argument un tipus de dades incorrecte, etc.) JavaScript no és fortament orientat a tipus: alguns errors que altres llenguatges capturen en compilar, ací només es mostren en executar	Resultats incorrectes en executar. Modificar el codi. Convé que en el codi es verifiquen sempre les restriccions a aplicar sobre els arguments de les funcions	> function suma(array) {return array.reduce(function(x,i){return x+i})} undefined > suma([1,2,3]) 6 > suma(1) TypeError: undefined is not a function at suma (repl:1.36) at repl:1:1 at REPLServer.defaultEval (repl.js:132:27) at bound (domain.js:254:14) at REPLServer.runBound [as eval] (domain.js:267:12) ....

Error Operacional	
Definició	Detecció/solució
Situacions excepcionals que poden sorgir durant el funcionament normal del programa. Poden deure's a: • l'entorn (ex. ens quedem sense memòria, massa fitxers oberts) • la configuració del sistema (ex. no hi ha una ruta cap a un host remot) • ús de la xarxa (ex. problemes amb l'ús de sockets) • problemes d'accés a un servei remot (ex. no puc connectar amb el servidor) • inconsistència en dades introduïdes per l'usuari • etc.	construccions try, catch, throw, similars a les de Java Estratègia: - Si és clar com resoldre l'error, gestiona'l (ex. error en obrir un fitxer per a escriptura -> crear-lo com a fitxer nou) - Si la gestió de l'error no és responsabilitat d'aquest fragment de programa, propagar l'error a l'invocador - Per a errors transitoris (ex. problemes amb la xarxa), reintentar l'operació Si no podem gestionar ni propagar l'error - si impedeix continuar amb el programa, avorta - en un altre cas, anota l'error en un log

Per a minimitzar els possibles errors, es recomana:

- utilitzar el mode stricte: **`node --use_strict fichero.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

PRIMERA PART

Execució de programes JavaScript

Adjunta, en la documentació associada a aquesta pràctica de laboratori, es troba la carpeta denominada **`javascript`** que conté diversos programes d'introducció als requeriments de l'assignatura.

- En aquests programes s'aborden algunes característiques bàsiques de JavaScript tals com: clàusula **this** (juntament amb la funció **bind**), clausures, programació asincrònica, etc.
- Pots veure la relació dels mateixos en l'apartat final d'aquest text.

L'activitat associada a aquesta part consisteix en l'anàlisi, estudi i execució dels codis abans esmentats, traient les pertinents conclusions.

Es deixen al professor els detalls explicatius addicionals, juntament amb la planificació i desenvolupament d'aquesta primera part.

SEGONA PART

Introducció a NodeJS

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.

- 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);
});
```

- 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");
});
```

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

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

(**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);
}
```

(**write2.js**)

```
//Escriptura asincrònica de fitxers

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

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

function cbEscritura(fitxer){
        console.log("escriptura realitzada en: "+fitxer);
}

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

Programació asincrònica: esdeveniments particularitzats programats

Ús del mòdul events. Analitza el codi següent i comprova el seu funcionament:

(**emisor1.js**)

```
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')
```

Analitza aquest altre exemple (emisor2.js). Quan generem esdeveniments es poden passar valors associats (arguments per a qui escolte i processe l'esdeveniment)

(**emisor2.js**)

```
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: Completa el fitxer emisor3.js per a complir les especificacions subsegüents.

(**`emisor3.js`**)

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

function rand() { // Ha de tornar valors aleatoris dins del rang [2000,5000) (ms)
   ... // Math.floor(x) torna la part entera del valor x
   ... // Math.random() torna un valor aleatori en el rang [0,1)
}

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

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

function etapa() {
   ...
}

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

ESPECIFICACIONS: Durant l'execució es completen diverses etapes, cadascuna amb una duració aleatòria entre 2 i 5 segons. En completar-se cada etapa:

- S'emetran els esdeveniments e1 i e2, passant com a valor associat el de la variable inc
- La variable inc augment una unitat
- En finalitzar cada etapa ha de mostrar-se per consola la seua duració.

Exemple d'execució (únicament el fragment inicial):

```
e1 --> 0
e2 --> 0
duracion 3043
e1 --> 1
e2 --> 01
duracion 3869
e1 --> 3
e2 --> 012
duracion 2072
e1 --> 6
e2 --> 0123
duracion 2025
e1 --> 10
e2 --> 01234
```

Interacció client/servidor

MÒDUL HTTP

Per a desenvolupament de servidors Web (servidors HTTP)

Ex.- servidor web (**`ejemploSencillo.js`**) que saluda al client que contacta amb ell

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

Com es tracta del primer cas d'un exercici que usa ports, has de recordar la possibilitat de conflicte. Si el teu DNI fóra, p. ex., 29332481 disposes dels ports 54810 a 54819. Pots substituir el número indicat en l'enunciat (8000) pel primer lliure a la teua disposició (54810 en aquest exemple).

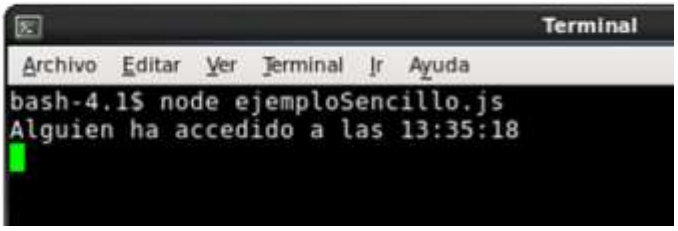
Executa el servidor i utilitza un navegador web com a client

- Accedeix a la URL **`http://localhost:8000`**

  (**`http://localhost:54810`** en l'exemple d'usuari comentat)

- Comprova en el navegador la resposta del servidor, i en la consola el missatge escrit pel servidor

Node y HTTP



MÒDUL NET

Client (`netClient.js`)	Servidor (`netServer.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');`
`  });`
	`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');`
	`  });`

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

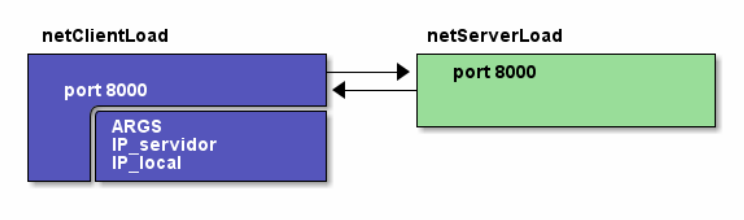
- **`process.argv.length`**: nombre d'arguments passats per la línia d'ordres
- **`process.argv[i]`** : permet obtenir l'argument i-èssim. Si hem usat l'ordre "**`node programa arg1`** …" llavors **`process.argv[0]`** conté la cadena '**`node`**', **`process.argv[1]`** conté la cadena '**`programa`**',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

Has de considerar que 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.

Consulta la càrrega de l'equip

En un entorn client/servidor a gran escala solen replicar-se els servidors (replicació horitzontal), i repartir el treball entre ells segons el seu nivell actual de càrrega.

Creem l'embrió per a un sistema d'aquest tipus, amb un únic client i un únic servidor (possiblement en màquines diferents) que es comuniquen a través del port 8000



- El servidor es denomina **`netServerLoad.js`**, i no rep arguments en línia d'ordres.
  - La següent funció **getLoad** calcula la seua càrrega actual (pots copiar el codi directament). Aquesta funció llig les dades del fitxer /proc/loadavg, 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)

```
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;
};
```

- El client es denomina **`netClientLoad.js`**: rep com a arguments en línia d'ordres l'adreça IP del servidor i la seua IP local
- 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**.

Per a facilitar les proves i depuració, s'ha deixat un servidor actiu en la IP **tsr1.dsic.upv.es**

Important:

- Has de basar-te en el codi de netClient.js i netServer.js descrits en el punt anterior
- Cal assegurar-se que tot procés finalitza (ex. amb `process.exit()`)
- Es pot esbrinar la IP des de la interfície web del portal, o usant l'ordre **ip addr**
- Completa tots dos programes, col·loca'ls en equips diferents col·laborant amb algun company, 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.

Intermediari entre 2 equips (proxy transparent)

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.

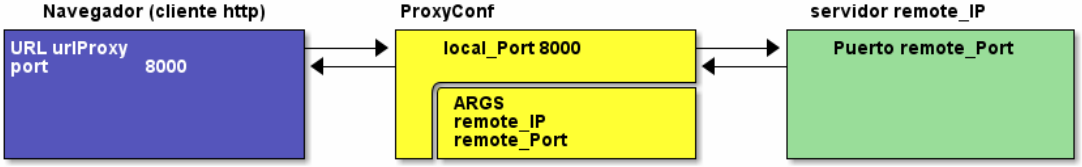
- Des del punt de vista del client, es tracta d'un servidor normal (oculta al servidor que realment completa el servei)
- Pot residir en una màquina diferent a la del client i la del servidor
- L'intermediari pot modificar ports i direccions, però no altera el cos de la petició ni de la resposta

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

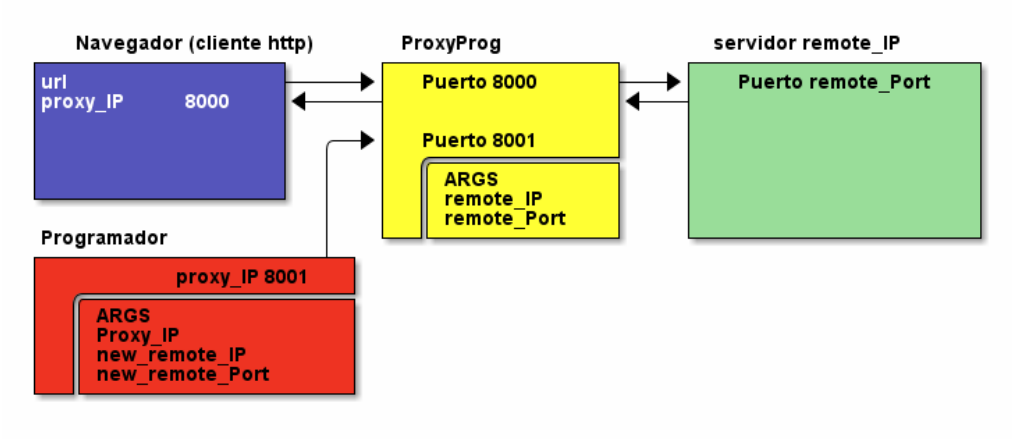
Plantegem tres passos incrementals:

1. Proxy bàsic (**Proxy**). Quan el client http (ex. navegador) contacta amb el proxy en el port 8000, el proxy redirigeix la petició al servidor web memex (158.42.186.55 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 resposta a missatges del programador en el port 8001



Important:

- Et proporcionem el codi de Proxy: analitza'l fins a comprendre el seu funcionament
  - Comprova que el seu funcionament és correcte usant un navegador web que utilitze http://adreça_del_proxy:8000/

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

- En l'escenari 2, hem de fer les següents proves:
  - Accés al servidor de la UPV (continuem utilitzant com a port local d'atenció de peticions el port 8000)
  - ProxyConf com a intermediari entre netClientLoad i netServerLoad
    - Si ProxyConf s'executa en la mateixa màquina que netServerLoad tenim una col·lisió en l'ús de ports -> modifica el codi de netServerLoad perquè use un altre port
    - 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 procés
- En l'escenari 3 és necessari codificar el client que actua com a programador
  - Aquest programa (**`programador.js`**) rep en línia d'ordres l'adreça IP del proxy, i els nous valors d'IP i port corresponents al servidor
  - El programa codifica aqueixos valors i els remet com a missatge al port 8001 del proxy, després de la qual cosa acaba
  - 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 de destinació els dos ja referenciats (memex.dsic.upv.es, IP 158.42.186.55, port 8080; i www.upv.es, IP 158.42.4.23, port 80) o qualsevol que utilitze HTTP (versió “no segura”), com www.aemet.es (IP 212.128.97.138, port 80).

REFERÈNCIES

1. **`Pràctica 0`**, en les zones de Recursos i Lliçons de l'assignatura en PoliformaT
2. **`Pràctica 1`**, també en la zona de Recursos i Lliçons de l'assignatura en PoliformaT. Inclou una part que conté els exemples de la carpeta **`JavaScript:`**

`j00.js`	Ús de funcions, objectes, clàusula this i funció bind().
`j01.js`	Declaració de variables. Ús de funcions i clausures.
`j02.js, j03.js`	Clausures de variables i funcions.
`j04.js, j05.js, j06.js, j07.js, j08.js, j09.js, j10.js, j11.js`	Ús d'operacions asincròniques, modelades amb la funció setTimeout.
`j12.js`	Exercici de recapitulació (operacions asincròniques i clausures)
`j13.js`	Execució de diverses operacions asincròniques paralelizadas a l'estil fork-join
`j14.js`	Execució de diverses operacions asincròniques serialitzades.