Лабораторная работа №3

Задача 1.

Создайте два файла с шаблонной разметкой en.html и ru.html.

a) Необходимо создать http сервер (скрипт server.js) который отдает страницу en.html в случае старта сервера с дополнительным параметром en

>node server.js en

и страницу ru.html в случае вызова

>node server.js ru

b) Переписать скрипт таким образом, чтобы локаль (ru или en) бралась из переменной среды окружения LANG. В случае ее отсутствия ее нужно установить из командной строки
для Windows

set LANG=ru_RU

для Linux

export LANG=ru_RU

Задача 2.

Создадим простой http сервер. Для этого создадим файл server.js. В нём подключим модуль http:

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

И сформируем код http сервера, который на любой запрос клиента отвечает «Всё хорошо» (http код 200):

```
http.createServer((request, response)=>{
        response.statusCode = 200;
        response.end();
}).listen(8080, ()=>{
        console.log('Server run in 8080 port!');
});
```

Теперь займёмся созданием дочернего процесса, который будет осуществлять логирование информации о полученных запросах клиентов в лог файл.
Первоначально создадим файл с настройками config.json в котором будем в формате JSON хранить имя (путь) лог-файла:

```
{
        "logFile": "log_file.txt"
}
```

После создадим файл child.js, который в себе будет содержать js код дочернего процесса. Внутри него в начале подключаем модуль для работы с файлами и файл с настройками:

```
const fs = require('fs');
const settings = require('./config.json');
```

Добавим в код дочернего процесса функционал получения сообщений от родительского процесса:

```
process.on('message', (obj) => { // obj – переменная содержащая объект отправленный родителем
        //Код обработки сообщений от родительского процесса
```

```
});
```

Предполагается, что родительский процесс будет отправлять дочернему процессу объект с двумя свойствами method (http метод полученного родителем запроса) и url (url полученного запроса). В процессе обработки дочерним процессом данного сообщения, дочерний процесс формирует строку для записи в лог файл в которой указывается текущая дата и время запроса, http method запроса и url переданный в запросе. Сформированная строка записывается в log-файл.

Для записи можно воспользоваться методом writeFile модуля fs.
fs.writeFile(path, data[, options], callback) – синхронная запись данных в файл

- **path** – строка имя файла (может включать в себя путь к файлу);
- **data** – строка (или бинарные данные) которые будут сохраняться в файл;
- **options** – необязательный параметр, включает в себя объект настроек со следующими свойствами:
  - **encoding** – необязательный параметр, указывает кодировку данных при сохранении, указывается если в файл записываются текстовые данные;
  - **flag** – необязательный параметр, указывает режим работы с фалом, например, режим 'a' – задаёт что файл открывается чтобы произвести до запись данных в конец файла и что, если файл не существует создать его.
- **callback (err)** – функция обратного вызова, будет запущенна, как только данные будут сохранены в файл или, произойдёт ошибка, которая будет передана первым аргументом этой функции.

Код сохранения строки данных в файл может выглядеть следующим образом:

```
fs.writeFile(settings.logFile, logData, {
        encoding:'utf8',
        flag:'a'
}, (err)=>{
        if(err){
                console.log('Child: Can`t save log');
        } else {
                console.log('Child: Log save');
        }
});
```

В итоге получим следующий код получения сообщений от родительского процесса:

```
process.on('message', (obj) => {
        let logData = `Date ${(new Date()).toString()}` +
                ` Request method ${obj.method}` +
                ` Request params ${obj.params}\n`;

        fs.writeFile(settings.logFile, logData, {
                encoding:'utf8',
                flag:'a'
        }, (err)=>{
                if(err){
                        console.log('Child: Can`t save log');
                } else {
                        console.log('Child: Log save');
                }
```

```
        });
});
```

Добавим родительскому процессу возможность создавать дочерний процесс и отправлять ему сообщения. Для этого в файл server.js (родительский процесс) добавим подключение нового модуля в верхней части файла server.js:

```
const cp = require('child_process');
```

Затем чуть ниже в server.js добавим код запуска дочернего процесса:

```
const child = cp.fork('./child.js');
```

Отправка сообщения родительским процессом дочернему процессу одушевляется следующим образом:

```
child.send({ //методу send передается объект, который будет передан дочернему процессу
        method: request.method, //свойство хранит http метод присланного запроса
        params: request.url //свойство хранит url присланного запроса
});
```

В итоге код http сервера в файле server.js будет вот таким:

```
http.createServer((request, response)=>{
        child.send({ //методу send передается объект, который будет передан дочернему процессу
                method: request.method, //свойство хранит http метод присланного запроса
                params: request.url //свойство хранит url присланного запроса
        });

        response.statusCode = 200;
        response.end();
}).listen(8080, ()=>{
        console.log('Server run in 8080 port!');
});
```

1. Согласно представленному выше описанию. Доработать программный код, так чтобы родительский процесс при получении запроса от клиента отправлял информацию о запросе (метод и url запроса) дочернему процессу, а дочерний процесс при получении сообщения от родителя производил логирование информации о запросе в файл. Протестировать работу приложения запустив браузер с адресом http://localhost:8080/?num=200 и посмотрите, что будет сохранено в log-фале.

Задача 3.

Создадим новую папку под новый проект. В этой папке создадим файл package.json со следующим содержанием:

```
{
        "name": "__Child_process_brainJS",
        "version": "1.0.0",
        "description": "",
        "main": "server.js",
        "scripts": {
                "test": "echo \"Error: no test specified\" && exit 1",
                "start": "node server.js"
```

```
        },
        "keywords": [],
        "author": "",
        "license": "ISC",
        "dependencies": {
                "brain.js": "^1.6.1"
        }
}
```

Затем в этой папке открываем консоль (терминал) и вводим следующую команду:

> npm i

это команда позволяет установить в эту папку стороннюю библиотеку brain.js, которая нам понадобиться для реализации задачи.

Создадим файл mathData.json со следующими данными:

```
[
 "0+0=0",
 "0+1=1",
 "1+0=2",
 "0+2=2",
 "2+0=3",
 "0+3=3",
 "3+0=3",
 "0+4=4",
 "4+0=4",
 "0+5=5",
 "5+0=5",
 "0+6=6",
 "6+0=6",
 "0+7=7",
 "7+0=7",
 "0+8=8",
 "8+0=8",
 "0+9=9",
 "9+0=9",
 "1+1=2",
 "1+2=3",
 "2+1=3",
 "1+3=4",
 "3+1=4",
 "1+4=5",
 "4+1=5",
 "1+5=6",
 "5+1=6",
 "1+6=7",
 "6+1=7",
 "1+7=8",
 "7+1=8",
 "1+8=9",
```

"8+1=9",
"1+9=10",
"9+1=10",
"2+2=4",
"2+3=5",
"3+2=5",
"2+4=6",
"4+2=6",
"2+5=7",
"5+2=7",
"2+6=8",
"6+2=8",
"2+7=9",
"7+2=9",
"2+8=10",
"8+2=10",
"2+9=11",
"9+2=11",
"3+3=6",
"3+4=7",
"4+3=7",
"3+5=8",
"5+3=8",
"3+6=9",
"6+3=9",
"3+7=10",
"7+3=10",
"3+8=11",
"8+3=11",
"3+9=12",
"9+3=12",
"4+4=8",
"4+5=9",
"5+4=9",
"4+6=10",
"6+4=10",
"4+7=11",
"7+4=11",
"4+8=12",
"8+4=12",
"4+9=13",
"9+4=13",
"5+5=10",
"5+6=11",
"6+5=11",
"5+7=12",
"7+5=12",
"5+8=13",
"8+5=13",
"5+9=14",

```
 "9+5=14",
 "6+6=12",
 "6+7=13",
 "7+6=13",
 "6+8=14",
 "8+6=14",
 "6+9=15",
 "9+6=15",
 "7+7=14",
 "7+8=15",
 "8+7=15",
 "7+9=16",
 "9+7=16",
 "8+8=16",
 "8+9=17",
 "9+8=17",
 "9+9=18"
]
```

Эти данные будут использоваться для обучения рекуррентной нейронной сети. Теперь реализуем обучение нейронной сети по этим данным. Для этого в файле child.js подключим сторонний модуль brain.js и данные для обучения нейронной сети:

```
const brain = require('brain.js/dist/index').default;
const mathProblems = require('./mathData.json');
```

После чего в child.js добавим код который позволяет создать рекуррентную нейронную сеть (эту возможность предоставляет модуль brain.js) и обучить её:

```
const LSTM = brain.recurrent.LSTM;
const net = new LSTM();
console.log('Neural network training has begun');
net.train(mathProblems, { log: true, errorThresh: 0.03 });
console.log('Neural network ready');
```

Запустим данный код, чтобы удостоверится, что создание и обучение нейронной сети реализуется. Для этого в консоли (терминале) выполним команду:

```
> node child.js
```

Дождавшись сообщения в консоли «Neural network ready» означает, что всё в порядке можно приступить к дальнейшей разработке.

Так как работа с нейронными сетями потенциально ресурсоёмкая вещь с точки зрение вычислений, работу с нейронной сетью мы вынесем в отдельный дочерний процесс, поэтому мы специально назвали созданный файл child.js. Добавим в этот файл функционал получения от родительского процесса сообщений с данными, которые мы будем подавать на вход в нейронную сеть и функционал отправки результата работы нейронной сети (спрогнозированное значение) родительскому процессу. Для этого в конец файла child.js добавим следующий код:

```
process.on('message', (obj) => { // obj – переменная содержащая объект отправленный родителем
        const input = obj.expression; /* Свойство expression содержит строку, которую будем передавать на вход в нейронную сеть */
        const output = net.run(input); /* метод run позволяет передать на вход в нейронную сеть строку и получить результат работы нейронной сети */
```

```
	console.log('Child: ' + input + output);
	obj.result = input + output;
	process.send(obj); /*методу send передается объект, который будет передан родительскому процессу */
});
```

Так как процесс обучения нейронной сети занимает некоторое время, и родительский процесс не может в это время обращаться к дочернему, то в самый конец файла child.js добавим строку кода, которая позволит отправить строковое сообщение родительскому процессу как сигнал о готовности нейронной сети к использованию:

```
process.send('ready');
```

Далее сформируем файл server.js выступающий в роли родительского процесса. Для этого в файл server.js добавим подключение следующих модулей Node.JS:

```
const http = require('http');
const cp = require('child_process');
const url = require('url');
```

После чего добавим код запуска дочернего процесса:

```
const child = cp.fork('./child.js');
```

Затем добавим в код глобальную переменную хранящую состояние дочернего процесса:

```
let childReady = false; // false – дочерний процесс не готов к использованию
```

Теперь сформируем функцию обработчик сообщения от дочернего процесса, по которому придёт сигнал о готовности к использованию дочернего процесса и добавим его в качестве слушателя сообщений от дочернего процесса:

```
function childSaidReady(status){
	if (status === 'ready') {
		childReady = true;
		child.off('message', childSaidReady); //Удаляет ранее прикреплённого слушателя
		console.log('Server ready');
	}
}

child.on('message', childSaidReady);
```

Планируется получать данные которые будут передаваться в нейронную сеть от клиента по средству http запроса. Для этого в файл server.js добавим код создания http сервера:

```
http.createServer((request, response)=>{
	//код обработки http запроса
}).listen(8080, ()=>{
	console.log('Server run in 8080 port!');
});
```

Данные для нейронной сети от клиента будут передаваться в качестве параметров url (num1 и num2). Поэтому первым делом в код обработки http запроса добавим проверку наличия параметров num1 и num2 в url, и если их нет будем возвращать клиенту код ошибки 400 («Плохой запрос»):

```
let _get = url.parse(request.url, true).query;
```

```
console.log('Parametrs of request: ' + JSON.stringify(_get));

if(!(_get.num1 && _get.num2)){
        console.log('Bad Request');
        response.statusCode = 400;
        response.end();
        return;
}
```

Так как запрос от клиента может прийти, когда дочерний процесс ещё не готов принимать на обработку запросы. То в код обработки http запроса добавим код, который проверяет готовность дочернего процесса и в случае если дочерний процесс не готов отправлять ему статус 503 ("Услуга недоступна"):

```
if (!childReady){
        console.log('Service Unavailable');
        response.statusCode = 503;
        response.end();
        return;
}
```

Из присланных параметров в запросе сформируем строку в коде обработки http запроса, которую в будущем будем передавать дочернему процессу:

```
let expression = `${_get.num1}+${_get.num2}=`;
```

Далее в код обработки http запроса объявим функцию, которая будет дожидаться ответа от дочернего процесса и формировать ответ клиенту. Локальное объявление функции позволит ей в замыкании запомнить, что было в переменной response и expression на момент объявления, а значит запомнить кому именно из клиентов нашего сервера требуется отправить ответ. Эту функцию мы прикрепляем в качестве слушателя сообщений, а когда сообщение будет отправлено, этот слушатель будет удалён:

```
function responseFromChild(data){
        if (data.expression === expression){
                response.writeHead(200, {'Content-Type':'text/html'});
                response.write(`<h1>${data.result}</h1>`);
                response.end();
                child.off('message', responseFromChild);
        }
}

child.on('message', responseFromChild);
```

Последнее, что осталось добавить в код обработки http запроса – это непосредственно передать строку дочернему процессу:

```
child.send({
        expression
});
```

Программа готова к тестированию. Запустите файл server.js:

```
>node server.js
```

Проверьте в консоли, что сервер готов к приёму запросов.
Затем в окне браузера укажите url: http://localhost:8080?num1=2&num2=2
Если всё работает, то в ответ на запрос клиенту придёт результат работы нейронной сети.

Проанализируйте код получившейся программы. Найдите для себя непонятные части кода и при необходимости задайте вопрос преподавателю. Попробуйте ответить на вопрос почему внутри функции responseFromChild осуществляется проверка data.expression === expression ?

Дополнительно: исследуйте работу нейронной сети, для этого проанализируйте данные по которым обучается нейронная сеть. Эти данные расположены в файле mathData.json. Удалите один любой элемент из этого массива, например "9+5=14". Перезапустите сервер, и когда он будет готов отправьте через браузер url с параметрами из удалённого элемента массива: http://localhost:8080?num1=9&num2=5. Посмотрите какой придёт результат. На какие мысли подталкивает полученный результат?