Teoria Współbieżności

Laboratorium 9 - Przetwarzanie asynchroniczne (wstęp do Node.js)

Magdalena Pastuła

Data laboratorium: 31.11.2020

1. Zadanie do wykonania.

Celem ćwiczenia było zapoznanie się z Node.js i wykonanie 2 ćwiczeń z jego wykorzystaniem.

2. Koncepcja rozwiązania.

2.1a

Zadanie to wykorzystuje mechanizm obietnic (Promises).

2.1b

Zadanie to wykorzystuje funkcję waterfall z biblioteki async.

2.2

Do zaimplementowania tego zadania wykorzystano bibliotekę glob, która pozwala na dopasowanie plików za pomocą wyrażeń regularnych dla całego drzewa plików. Za jej pomocą możliwe jest pobranie listy wszystkich plików w danym katalogu, włączając w to pliki z podkatalogów.

3. Implementacja i wyniki.

3.1a

Implementacja zadanej funkcji loop:

```
function loop(n) {
    console.log('running for: ' + n)
    if (n !== 0) {
        work().then(() => loop(n-1))
    }
}
```

Po wykonaniu programu dostajemy następujące komunikaty:

```
running for: 4
1
task 1 done
2
```

```
task 2 done
3
task 3 done
done
running for: 3
1
task 1 done
2
task 2 done
3
task 3 done
done
running for: 2
1
task 1 done
2
task 2 done
3
task 3 done
done
running for: 1
1
task 1 done
2
task 2 done
3
task 3 done
done
running for: 0
```

3.1b

Implementacja zadanej funkcji loop:

```
function loop(n) {
    const executions = []
    for(let i = 0; i< n; i++){
        executions.push((callback) => work().then(() => callback()))
    }
    async.waterfall(executions, (a, b) => console.log('done'))
}
```

Po wykonaniu programu dostajemy następujące komunikaty:

```
1
task 1 done
2
task 2 done
3
```

```
task 3 done
done
1
task 1 done
2
task 2 done
3
task 3 done
done
1
task 1 done
2
task 2 done
3
task 3 done
done
1
task 1 done
2
task 2 done
3
task 3 done
done
done
```

3.2 Wersja asynchroniczna.

Poniżej znajduje się implementacja wersji asynchronicznej

```
var fs = require('fs')
var glob = require('glob')

var count = 0
var done = false

const files = glob.sync('./PAM08/**/*')
const filesIterator = files.entries()
console.log(files.length)

function readFromFile(file, onEnd) {
    if (fs.lstatSync(file).isDirectory()) {
        onEnd()
        return
    }
    fs.createReadStream(file).on('data', function (chunk) {
        count += chunk.toString('utf8')
            .split(/\r\n|[\n\r\u0085\u2028\u2029]/g)
            .length - 1;
    }).on('end', function () {
        // console.log(file, count);
        onEnd()
```

```
    }).on('error', function (err) {
        console.error(err);
        onEnd()
    });
}

function work() {
    const next = filesIterator.next()
    if (!next.done) {
        readFromFile(next.value[1], work)
    } else {
        done = true
    }
}

work()
```

3.2 Wersja synchroniczna.

Poniżej znajduje się implementacja wersji synchronicznej

```
//find . -name '*' | xargs wc -l | sort -nr
var fs = require('fs')
var glob = require('glob')
var fsPromises = require('fs').promises
var lodash = require('lodash')
var async = require("async");

var count = 0

async function readFromFile(file) {
    return new Promise((resolve, reject) => {
        fsPromises.lstat(file).then((lstat) => {
            if (lstat.isDirectory()) {
                resolve()
            } else {
                fs.createReadStream(file).on('data', function (chunk) {
                    count += chunk.toString('utf8')
                        .split(/\r\n|[\n\r\u0085\u2028\u2029]/g)
                        .length - 1;
                }).on('end', function () {
                    // console.log(file, count);
                    resolve()
                }).on('error', function (err) {
                    console.error(err);
                    reject()
                });
            }
        })
    })
}
```

```
glob('./PAM08/**/*', (err, files) => {
    console.log(files.length)
    const executions = []
    for(let file of files){
        executions.push((callback) => readFromFile(file).then(() => callback()))
    }
    //Limited version due to the amount of files that the os is capable of opening
    async.parallelLimit(executions, 100, (a, b) => console.log('done'))
})
```

Obydwie wersje zwracają wartość 1050 jako liczbę lini w podanym jako przykład folderze. Natomiast czas wykonania wersji synchronicznej wyniósł 412.229, a asynchronicznej 203.858 ms. Dla większego folderu z łączną liczbą linijek równą 13 019 czasy te wyniosły odpowiednio 5028.463 ms oraz 2391.378 ms.

4. Wnioski z ćwiczenia.

Już na pierwszy rzut oka widać, że wersja asynchroniczna programu jest blisko dwa razy szybsza niż wersja synchroniczna. Prawdopodobnie jest to kwestia dostępu do pliku: w momencie czekania na dostęp wersja asynchroniczna czyta z innego pliku, a wersja synchroniczna nic innego nie robi.

5. Bibliografia.

1. Przetwarzanie asynchroniczne
2. Mechanizm działania pętli zdarzeń
3. Biblioteka async
4. Biblioteka glob
5. Pomiar czasu wykonania w JS