Niti

Prilikom surfovanja webom možda ste nekada bili svedoci kratkotrajnog zamrzavanja korisničkog okruženja stranice koju pregledate. Takvo kratkotrajno zamrzavanje često je posledica neke blokirajuće operacije, do koje dolazi prilikom izvršavanja JavaScript koda. S obzirom na to da se, podrazumevano, JavaScript kod izvršava naredbu po naredbu, može se dogoditi da neka vremenski zahtevna operacija blokira kompletan sajt na određeni vremenski period. Kako biste razumeli situacije u kojima može doći do ovakve pojave, ali i načine na koje se takvi problemi prevazilaze, u lekcijama ovoga modula biće reči o načinima za postizanje višenitnog i asinhronog izvršavanja. Prva lekcija ovoga modula biće posvećena pojmu višenitnog izvršavanja.

Primeri blokirajuće operacije

Primer koji će biti prikazan u nastavku ilustrovaće upravo spomenutu situaciju, u kojoj dolazi do zamrzavanja kompletne stranice usled izvršavanja vremenski zahtevne operacije:

```
        <div id="my-button">Click me</div>

        <script>
            var myButton = document.getElementById("my-button");

            myButton.addEventListener("click", function() {

                //time-intensive operation start

                var data = [];
                for ( let i = 0; i < 50000; i++) {
                    let random = Math.random().toString().split("0");
                    data = data.concat(random);
                }
                console.log(data.length);

                //time-intensive operation end

                console.log("Hello from statement after time-intensive
operation.");

            });
        </script>
```

Primer definiše jedan HTML element na koji korisnik treba da klikne kako bi se aktivirala dodatna logika. Reč je o `div` elementu, čiji je izgled definisan upotrebom sledeće stilizacije:

```
            #my-button {
                width: 200px;
                height: 40px;
                color: white;
                background-color: teal;
                border-radius: 4px;
                line-height: 40px;
                text-align: center;
                font-family: sans-serif;
            }
```

```
#my-button:hover{
    background-color: rgb(30, 109, 109);
    cursor: pointer;
    font-weight: bold;
}
```

Na ovaj način se dobija `div` element kao na animaciji 7.1.

Animacija 7.1. Izgled i ponašanje dugmeta koje je kreirano upravo prikazanim kodom

Ovakav specijalan efekat, koji se dobija prelaskom pokazivača miša preko elementa, dodat je kako biste na lakši način mogli da uvidite zamrzavanje kompletne stranice, do čega dolazi prilikom izvršavanja vremenski zahtevne JavaScript operacije.

Vremenski zahtevna operacija može se videti unutar `script` elementa i funkcije koja se aktivira prilikom klika na `div`. Ona je uokvirena komentarima *time-intensive operation start* i *time-intensive operation end*, kako biste na lakši način mogli da vidite njen početak i kraj. Vremenski zahtevna operacija simulira se jednom petljom sa 50 hiljada iteracija. Unutar svake iteracije, generiše se nasumični broj, on se konvertuje u `string`, a zatim se takav `string` deli na svim onim mestima gde se pojavljuje karakter 0. Deobom teksta dobija se niz, a nizovi iz svake iteracije se objedinjuju unutar promenljive `data`. Nakon završetka for petlje, štampa se dužina kreiranog `data` niza. Ovakva logika nema preterano smisla u praksi, a ovde je kreirana samo kako bi se simulirala vremenski zahtevna operacija.

Nakon vremenski zahtevne operacije, definisana je još jedna naredba – naredba za ispis poruke unutar konzole. Kada ovakav HTML dokument otvorite unutar pregledača i izvršite klik na kreirano dugme, moći ćete da primetite da se kompletna stranica zamrzava na određeni vremenski period, odnosno sve dok se ne završi vremenski zahtevna operacija. Zamrzavanje je najlakše primetiti na kreiranom dugmetu – efekat koji se dobijao prelaskom kursora miša ne funkcioniše više, a sam kursor i dugme su zamrznuti u stanju klika.

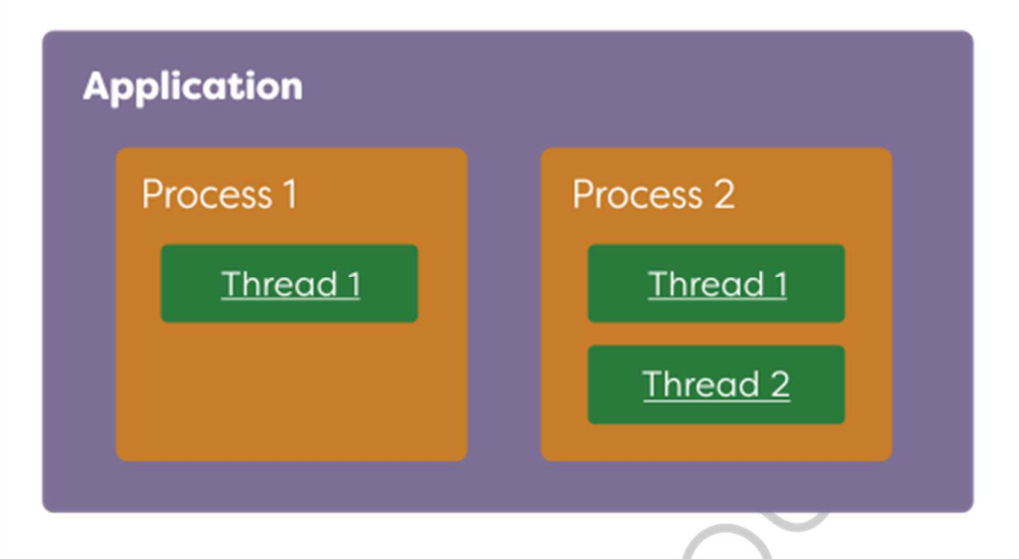
Praćenjem teksta koji se ispisuje unutar konzole može se videti da se ni naredba koja je definisana nakon vremenski zahtevne operacije ne izvršava, sve dok se takva operacija ne završi. Onoga trenutka kada se vremenski zahtevna operacija završi, prelazi se na izvršavanje naredne naredbe.

Vremenski zahtevna operacija iz prikazanog primera se veoma često naziva blokirajući kod, s obzirom na to da onemogućava izvršavanje programske logike koja je definisana nakon nje.

Zbog čega upravo prikazani primer blokira izvršavanje JavaScript koda?

Kako biste razumeli zbog čega upravo prikazani primer blokira izvršavanje JavaScript koda koji se nalazi nakon blokirajuće operacije, prethodno je neophodno razumeti na koji način se obavlja izvršavanje programa, odnosno aplikacija od strane hardvera, posredstvom operativnog sistema kompjutera.

U očima operativnog sistema, jedna aplikacija, odnosno program, može biti sačinjen iz više procesa. Pri tome, svaki proces unutar sebe može imati proizvoljan broj niti (slika 7.1).



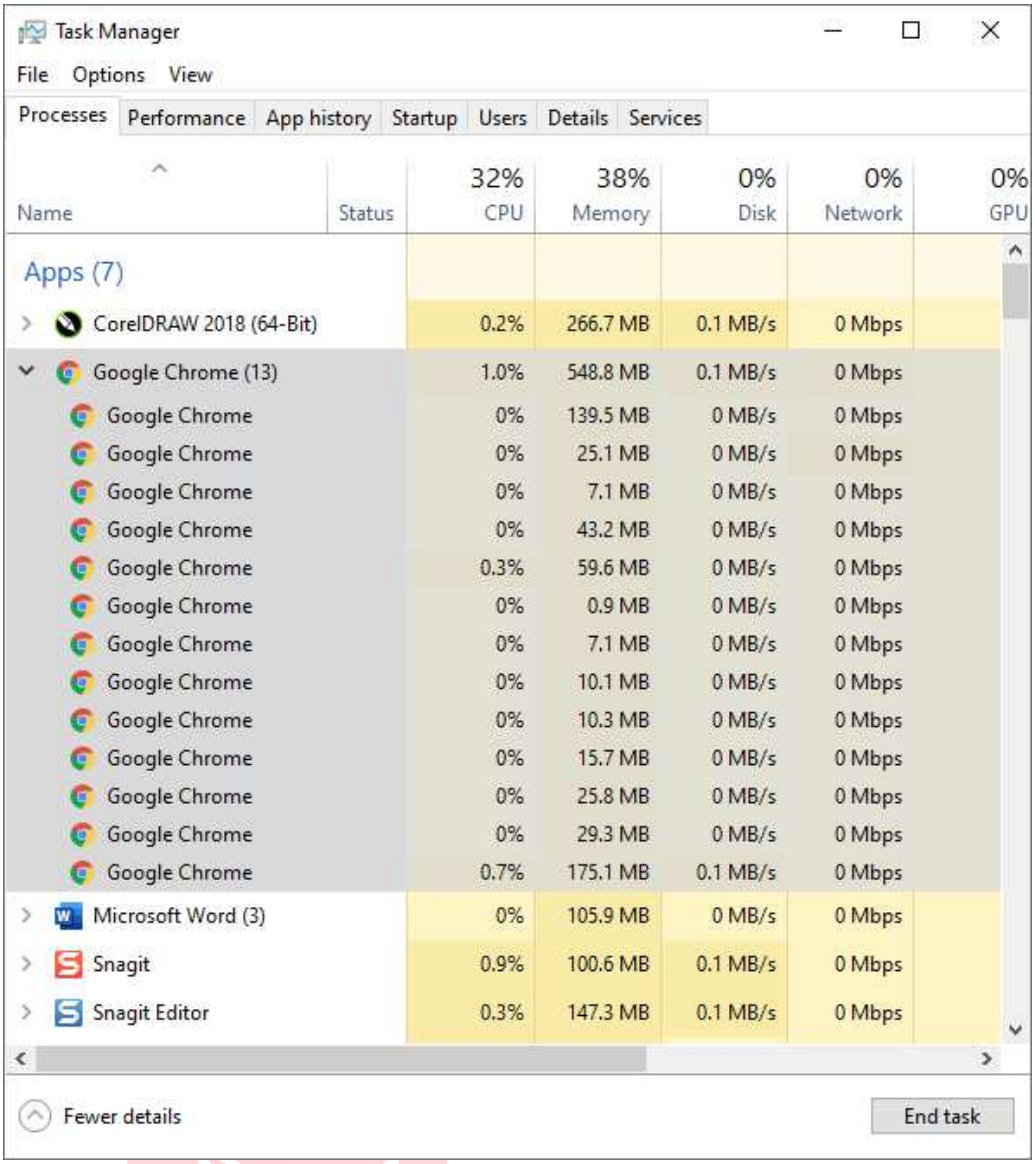
Slika 7.1. Procesi i niti

Na slici 7.1. može se videti kako jedna aplikacija izgleda iz ugla operativnog sistema na kom se izvršava. Aplikacija poseduje dva procesa. Prvi proces poseduje jednu nit (*engl. thread*), a drugi proces dve niti.

Procesi su način na koji operativni sistem izoluje i razdvaja izvršavanje programa. Uprošćeno se može reći da su procesi zapravo programi ili njihovi delovi koji se posredstvom operativnog sistema izvršavaju na nekom kompjuteru.

Nit je najmanja izvršna jedinica koja može postojati unutar procesa. Svaki proces sastoji se iz jedne ili više niti. Procesi poseduju minimalno jednu nit, koja se kreira prilikom nastanka procesa i koja se drugačija naziva primarna nit. Jedno jezgro kompjuterskog procesora u jednom trenutku može da izvršava samo jednu nit.

Iz ugla frontend programiranja, web pregledači (Chrome, Firefox, Safari...) su primeri aplikacija, odnosno programa koji se direktno izvršavaju na operativnim sistemima klijentskih uređaja. Većina modernih web pregledača za svaki otvoreni tab kreira zasebni proces. Na primer, na operativnom sistemu Windows možemo se u to veoma lako uveriti (slika 7.2).



Slika 7.2. Web pregledač Chrome sa 13 otvorenih tabova – za svaki tab je kreiran zaseban proces

Pitanje

Jedna aplikacija mora imati minimalno dve niti.

- Tačno.
- **Netačno.**

Objašnjenje:

Aplikacija mora imati minimalno jednu nit. Ona se veoma često naziva glavna nit aplikacije. Postojanje dodatnih niti nije obavezno.

Na slici 7.2. može se videti da Chrome za svaki otvoreni tab kreira zasebni proces. Svaki od takvih procesa podrazumevano poseduje po jednu nit, koja se koristi za izvršavanje JavaScript programskog koda koji samostalno pišemo. To praktično znači da se naš JavaScript kod podrazumevano izvršava unutar jedne niti. Unutar svake niti, kod se podrazumevano izvršava linijski, odnosno naredbu po naredbu. Drugim rečima, kako bi jedna naredba započela izvršavanje, prethodna se mora u potpunosti završiti. Stoga, u slučaju izvršavanja vremenski zahtevne operacije, kao što je to učinjeno u nešto ranije prikazanom primeru, kompletna stranica postaje zamrznuta. Rešenje ovakvog problema može se postići korišćenjem posebnog načina izvršavanja programskog koda – višenitnog izvršavanja.

Slika 7.3. Uporedni prikaz izvršavanja koda u jednoj i više niti

Slika 7.3. ilustruje uporedni prikaz izvršavanja programskog koda u jednoj i više niti. Gornja polovina slike ilustruje izvršavanje koda u jednoj niti. Bitno je primetiti da metode `doSomeWork2()` i `doSomeWork3()` moraju da čekaju završetak metode `doSomeWork1()`.

Donja polovina slike 7.3. ilustruje višenitno izvršavanje. Na slici se mogu videti dve niti. Vremenski zahtevna operacija `doSomeWork1()` prebačena je u zasebnu nit. Stoga, metode `doSomeWork2()` i `doSomeWork3()` ne moraju da čekaju, već se izvršavaju uporedo sa metodom `doSomeWork1()`.

Dugo vremena u JavaScriptu nije postojao način da se pored glavne niti kreira još neka dodatna nit, unutar koje bi mogla da se prebaci vremenski zahtevna operacija, te na taj način rastereti glavna nit i spreči zamrzavanje kompletne stranice. Ipak, tako nešto postalo je moguće uvođenjem jednog posebnog aplikativnog programskog interfejsa – Web Workers.

Web Workers API

Web Workers je skup funkcionalnosti koje u JavaScriptu obezbeđuju višenitno programiranje (*engl. multithreaded programming*). Stoga, mi sada možemo iskoristiti Web Workers skup funkcionalnosti za modifikaciju nešto ranije prikazanog primera, koji je za posledicu imao zamrzavanje kompletne stranice.

Web Workers skupu funkcionalnosti pristupa se korišćenjem objekta `Worker`. Ovaj objekat je moguće kreirati na sledeći način:

```
const worker = new Worker('time-intensive.js');
```

Prilikom kreiranja objekta `Worker`, njemu se prosleđuje putanja do fajla sa JavaScript kodom koji je potrebno izvršiti u zasebnoj niti, koja se drugačije naziva radna nit (*engl. worker thread*). Tako dolazimo do prve osobenosti Web Workers API-ja – u zasebnoj niti može se izvršiti samo JavaScript kod koji je smešten unutar zasebnog fajla.

Kod koji se nalazi u zasebnom fajlu (u našem slučaju je to `time-intensive.js` fajl) izvršiće se unutar zasebne niti. Ipak, njegovo izvršavanje je na neki način potrebno pokrenuti i nakon izvršavanja dobiti povratnu vrednosti. Celokupna komunikacija između koda koji se nalazi unutar glavne i onog unutar radne niti, obavlja se razmenom poruka. Stoga, kako bi se logika unutar radne niti pokrenula, njoj je dovoljno uputiti poruku, pozivanjem metode `postMessage()` nad kreiranim `Worker` objektom:

```
worker.postMessage('Start!');
```

Unutar zagrada se navode parametri koji će biti prosleđeni radnoj niti. Naša funkcionalnost ne zahteva bilo kakav parametar, stoga je u primeru metodi `postMessage()` prosleđena simbolična poruka *Start!*, koja se neće koristiti.

Unutar radne niti, obrada upućenih poruka obavlja se definisanjem funkcije kao vrednosti `onmessage` promenljive:

```
onmessage = function () {

    var data = [];
    for (let i = 0; i < 50000; i++) {
        let random = Math.random().toString().split("0");
        data = data.concat(random);
    }

    postMessage(data);
}
```

Ovakva anonimna funkcija biće aktivirana kada se iz glavne niti pošalje radnoj niti poruka upotrebom metode `postMessage()`. Unutar anonimne funkcije, definisana je vremenski zahtevna logika, koja je dosad bila smeštana unutar glavne niti.

Nakon završetka vremenski zahtevne operacije unutar radne niti, rezultat obrade je na neki način potrebno vratiti glavnoj niti. I tako nešto se postiže upućivanjem poruke pozivanjem metode `postMessage()`. U prikazanom primeru, ovakvoj metodi prosleđena je vrednost promenljive `data`.

Na kraju, u glavnoj niti se poruka koja sadrži povratnu vrednost iz radne niti obrađuje registrovanjem funkcije za obradu događaja `onmessage`:

```
worker.onmessage = function (e) {
    //time-intensive operation end
    console.log(e.data.length);
}
```

Kompletan kod preuređenog primera izgleda ovako:

index.html:

```
<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>JavaScript Functions</title>
        <style>
            #my-button {
                width: 200px;
                height: 40px;
                color: white;
                background-color: teal;
                border-radius: 4px;
                line-height: 40px;
                text-align: center;
                font-family: sans-serif;
            }

            #my-button:hover {
                background-color: rgb(30, 109, 109);
                cursor: pointer;
                font-weight: bold;
            }
        </style>
    </head>

    <body>

        <div id="my-button">Click me</div>

        <script>
            var myButton = document.getElementById("my-button");
            const worker = new Worker('time-intensive.js');

            myButton.addEventListener("click", function () {

                //time-intensive operation start
                worker.postMessage('Start!');

                console.log("Hello from statement after time-intensive
operation.");

                worker.onmessage = function (e) {
                    //time-intensive operation end
                    console.log(e.data.length);
                }

            });

        </script>

    </body>

</html>
```

time-intensive.js

```
onmessage = function () {

    var data = [];
    for (let i = 0; i < 50000; i++) {
        let random = Math.random().toString().split("0");
        data = data.concat(random);
    }

    postMessage(data);
}
```

Napomena za pokretanje primera

Pokretanje upravo prikazanog primera nije moguće postići upotrebom file:// protokola. Naime, većina web pregledača poseduje sigurnosnu zaštitu, koja stranicama ne dozvoljava direktan pristup fajlovima koji se nalaze na fajl sistemu korisnika. Stoga, za uspešno funkcionisanje prikazanog primera neophodno je njegove fajlove smestiti na neki HTTP server, bilo lokalni (wamp, xamp) ili udaljeni.

Pokretanjem upravo modifikovanog primera može se videti da vremenski zahtevna operacija više ne blokira kompletnu stranicu. Unutar konzole prvo se dobija poruka od naredbe koja se nalazi nakon vremenski zahtevne operacije. Onoga trenutka kada se vremenski zahtevna operacija iz zasebne niti završi, unutar konzole se prikazuje i rezultat njene obrade.

Rezime

- blokirajuća operacija je vremenski zahtevna operacija, koja ne dozvoljava narednim operacijama da se izvrše sve dok ona ne završi svoju logiku;
- iz ugla operativnih sistema, aplikacije se posmatraju kao procesi;
- procesi su način na koji operativni sistem izoluje i razdvaja izvršavanje programa;
- svaka aplikacija može biti sačinjena iz jednog ili više procesa;
- svaki proces može imati jednu ili više niti;
- nit je najmanja izvršna jedinica koja može postojati unutar procesa;
- moderni web pregledači najčešće za svaki otvoreni tab kreiraju zasebni proces;
- JavaScript kod koji mi pišemo izvršava se unutar jedne niti;
- višenitno izvršavanje podrazumeva izvršavanje koda u većem broju niti;
- Web Workers je skup funkcionalnosti koje u JavaScriptu obezbeđuju višenitno programiranje;
- dodatne niti koje se kreiraju korišćenjem Web Workers API-ja zovu se radne niti;
- unutar radnih niti Web Workers API-ja može se izvršavati samo JavaScript kod koji je definisan u zasebnom fajlu.