Internet Programming

Week 8

Instructor:
Daniel Slack, P. Eng.
Applied Computer Science
University of Winnipeg

Web Workers

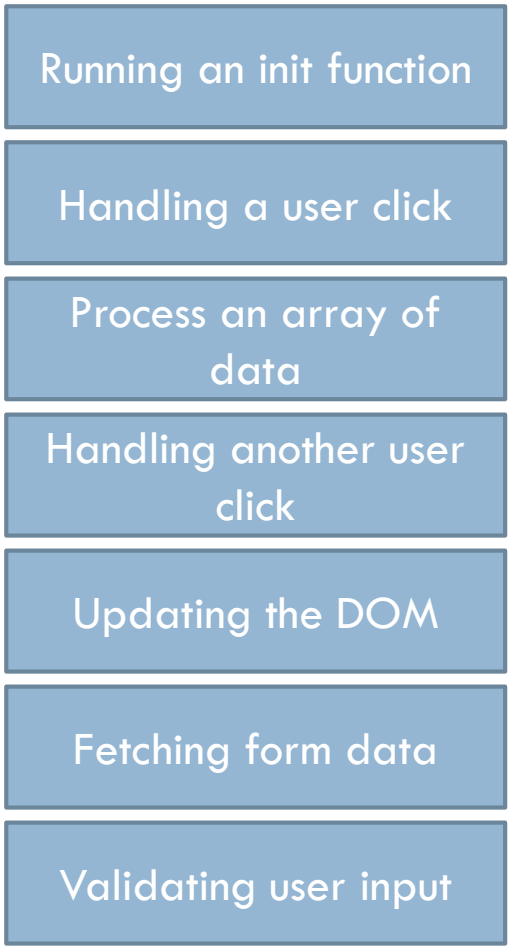
Web Workers

- Ever get a slow script message?
  - Q) How could a script be slow running with a multicore processors in your machine?
  - A) JavaScript can only do one thing at a time
    - Need a way to spawn worker threads
- HTML5 and Web Workers allows one to spawn workers to get more done

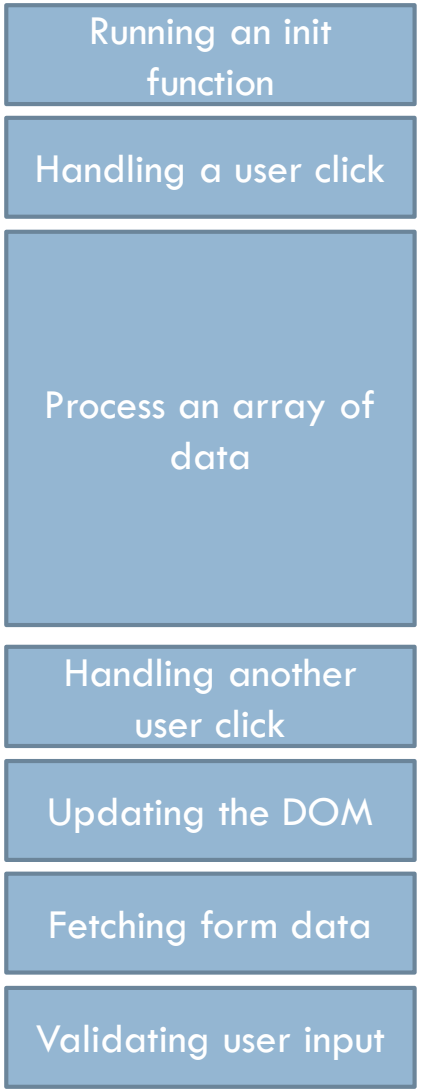
Single Threaded

- JavaScript only does 1 thing at a time
  - Called single threaded
  - Based on the idea of processes and threads
- Improves program performance
  - Writing programs with multiple threads can be challenging
- Disadvantage:
  - Tasks with high computational complexity can hijack your web application
  - *i.e.* complex tasks can leave your UI unresponsive

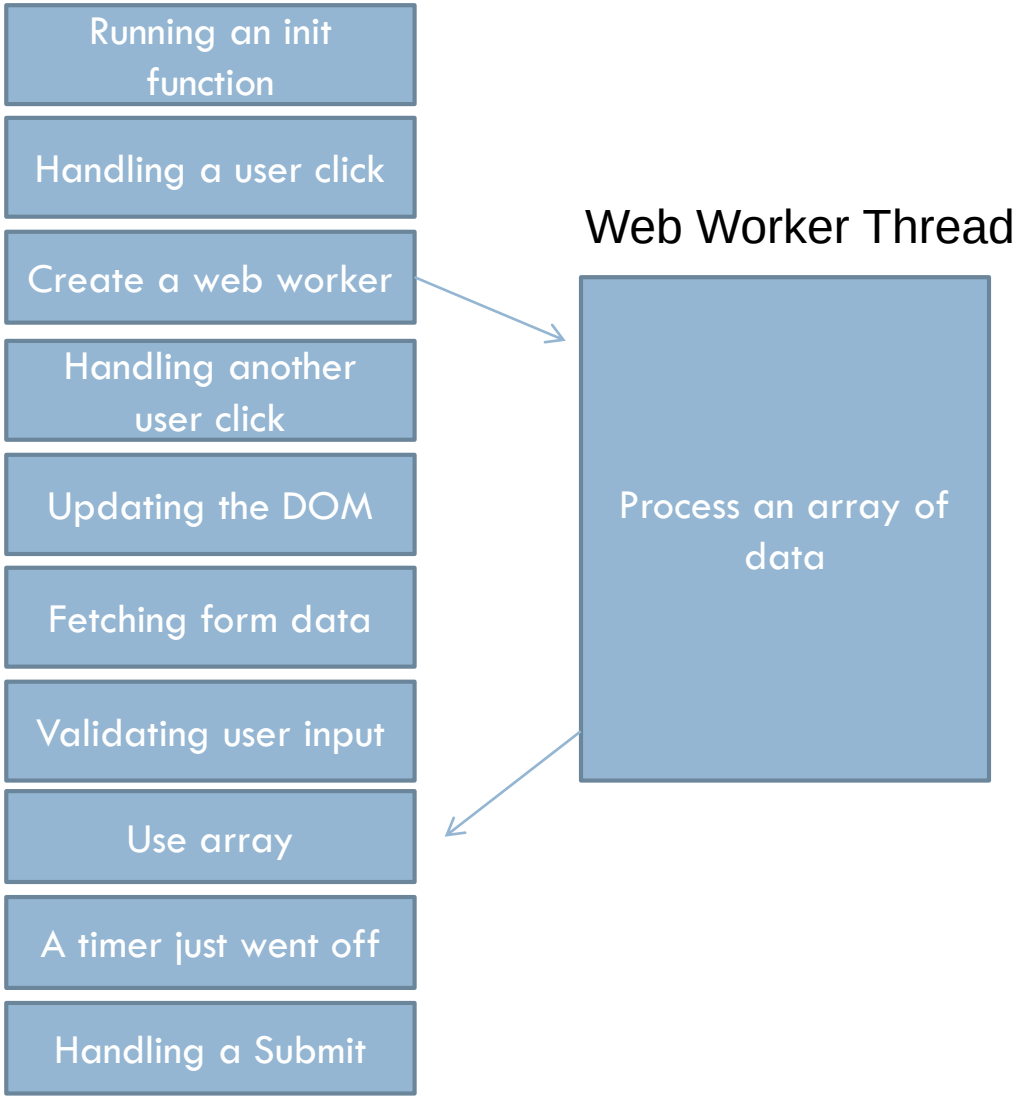
Typical JavaScript Thread



Single Threaded Problems



Multi-threaded Solution



Single Threaded

- How can JavaScript be single threaded?
  - What happens when a timer goes off?
- When an event occurs, that event is added to a queue
  - The JavaScript engine does not get to it until it is finished with its current task
- Web Workers allow you to offload the computationally complex operations to another thread

Web Workers

- Browser creates one or more workers
- Each worker is defined with its own JavaScript file
  - Contains all the code, or references to the code
- Workers are very restricted
  - They don't have access to many of the runtime objects available to the browser
    - e.g. The DOM or any variables or functions in your main code

Web Workers

- Browser sends a message to the worker to start it working
- Worker code:
  - Receives the message
  - Looks to see if there are any special instructions
  - Starts working
- Worker sends a message to indicate it has finished its task
  - Contains the final results
- Main browser incorporates results into page in some way

Web Workers

- Why not allow workers to access the DOM?
  - JavaScript engines highly optimize DOM operations based on the assumption that only one thread has access to the DOM
  - Otherwise, race conditions can occur
    - Seriously impact performance

Web Workers

- To create a new worker, you must create a worker object

```
var worker = new Worker("workerJSfile.js");
```

- You can create as many workers as you like
- Recall, to start the worker thread, you must send it a message

```
worker.postMessage(wkrMSGstr)
```

Web Workers

- What can you send in your messages?
  - Strings
    - “ping”
  - Array
    - [1, 2, 3, 5, 11]
  - JSON objects
    - {“message”: “ping”, “count”: 5);
- You cannot send functions
  - It might contain a reference to the DOM

Web Workers

- We must set up the main thread to receive the output of a worker
- We need to define a handler to receive a message from a web worker
  - Accomplished by defining the worker's onmessage property
  - In other words, define a function that will get called when a message is received from a worker

Web Workers

- The message from the worker is wrapped in an event object
- The event object's data property contains the message data
- The event object's target property contains a reference to the worker that sent the message
  - Useful if you need to know which worker sent the message

Web Workers

- Writing the worker JavaScript
  - First thing to do is ensure the Web Worker can receive messages from the main thread
    - Similar to window.onload property we have seen so often
  - We are going to set the onmessage handler for the worker itself
  - Every worker is ready to receive messages
    - You just have to give the worker a handler to process them

Example 01

- Simple Web Worker example

Web Workers: Misc.

- Web workers can access localStorage and make XMLHttpRequests
- Web workers have a global function named importScripts
  - Used to import one or more JavaScript files into your worker
  - When evoked, each js URL is retrieved and evaluated in order

```
importScripts(“http://bigscience.org/nuclear.js”,
“http://nasa.gov/rocket.js”,
“mylibs/atomsmasher.js”);
```

Web Workers: Misc.

- Is there a limit to the number of workers?
  - Yes, Web Workers are not intended to be created in large numbers
    - Require extra memory, and an OS thread
    - Costly in start-up time and resources
  - In general, create a limited number of them and assign them more work
    - Rather than creating many with a small amount of work
  - Aim for the number of logical cores in your system
    - Or, in the system of a common user for your site

Terminating Workers

- Workers take up memory in the browser
- They should be terminated once they are done their task and no longer necessary

```
worker.terminate();
```

- Caution: The worker script will abort if the worker still happens to be running
  - Be careful

Error Handling in Workers

- Errors happen in workers too
- Remember the Chrome debugger is very helpful
- You can also use the following onerror handler

```
worker.onerror = function(error) {
   document.getElementById(“output”).innerHTML =
      “There was an error in “ + error.filename _
      “ at line number “ + error.lineno  +
      “: “ + error.message;
}
```

ImportScripts and JSONP Request

- Script elements cannot be inserted into workers to make JSONP requests
- However, importScripts can be used to make JSONP requests

```
function makeServerRequest() {
   importScripts("http://SomeServer.com?callback=handleRequest");
}
function handleRequest(response) {
   postMessage(response);
}
makeServerRequest();
```

setInterval in Workers

- You can use setInterval (and setTimeout) in your workers to do the same task repeatedly

```
var quotes = [“I hope life isn’t a joke, because I don’t get it.”,
   “There is a light at the end of the tunnel…just pray it’s not a train!”,
   “Do you believe in love at first sight or should I walk by again?”];
function postAQuote() {
   var index = Math.floor(Math.random() * quotes.length);
   postMessage(quotes[index]);
}
postAQuote();
setInterval(postAQuote, 3000);
```

Subworkers

- Workers can create workers!
  - Yikes!
- Example:
  - Workers are given a region of the image to process
  - Each worker can decide if it should spawn another work if the region is bigger than a set size
- Created the same way
- Remember:
  - Workers are fairly heavy-weight
    - Take up memory and run as separate threads
  - Be cautious about how many you create

Web Workers By Example

- The Mandelbrot Set is a mathematical graph that is an infinitely long line on a finite surface
- Mandelbrot Set Example
- http://www.youtube.com/watch?v=MAzYWM7Yf4U
- http://www.youtube.com/watch?v=0jGaio87u3A
- https://www.youtube.com/watch?v=ohzJV980PIQ
- Advantages for this example
  - Generated by a very simple equation
  - Generating this set is computationally complex
    - Great for an example on Web Workers

Computing a Mandelbrot Set

- Textbook (Head First HTML5 Programming) has provided the code for calculating Mandelbrot pixel values
- We're interested in the general approach to performing the calculation
  - In order to parallelize the problem

Computing a Mandelbrot Set

- Big Picture

```
for (var i=0; i<numberOfRows; i++) {
   var row = computeRow(i);
   drawRow(row);
}
```

- We loop over each row of the image
- For each row, compute the pixels for the row
- Draw each row on the screen

Computing a Mandelbrot Set

- In practice, there are a few more details
  - Need to know width of the row, zoom factor, numerical resolution

Computing a Mandelbrot Set

- Parallelization of Code
  - Trick is to rework approach to divide problem up among a number of workers
    - Including handles for dealing with the results

```
for (var i=0; i<numberOfRows; i++) {
   var taskForRow = createTaskForRow(i);
   var row = computeRow(taskForRow);
   drawRow(row);
}
```

Multiple Workers

- We know how to create new workers
- How can they be used to do something complicated?
  - Like this example?
- Answer:
  - Break up the job into small independent tasks
    - Each worker is assigned an independent task
- Note:
  - The pattern given here can be applied to any problem that is parallelizable

Multiple Workers

- How does multiple workers improve the application?
  1. Applications that contain a lot of computing must still be responsive to the user
  2. Almost all systems these days ship with multicore processors
     - Multiple threads can run in parallel on these cores
     - System takes care of the details

Computing a Mandelbrot Set

Example 02

- Setting up the workers to draw the initial image

Example 03

- Handling a click event

Example 04

- Fitting the canvas to the browser window
  - Fractal image to fill the browser window
  - Resize the canvas

Animation

Animation

- In our animation, we used setTimeout (setInterval)
  - Whether we change element css
    - Margins, padding, positions
  - Or redraw on the canvas
- Problems?
  - Need to specify framerate
    - 60fps = 1000ms/60frames ~ 17
    - setTimeout(fn, 17);
  - Event loop consideration

Animation

- Event Loop
  - When we do some operation
    - Events, setTimeout
  - Message added to event loop
    - Represented as a queue
  - With a delay (like in setTimeout), other messages "queue jump"
  - `setTimeout(fn, 0)` pushes message onto queue
    - Existing messages finish, then "new" messages start
- Animation are causing repaints on-demand

Animation

- So, if not setTimeout, then what?
- window.requestAnimationFrame(fn)
  - Schedules the function ONLY when next repaint happens
  - Designed for animation, better handling of animation changes