Web Workers

Cpt S 489

Washington State University

Web Workers

- A lot like threads
- MDN page states:

“Web Workers are a mechanism by which a script operation can be made to run in a background thread separate from the main execution thread of a web application. The advantage of this is that laborious processing can be performed in a separate thread, allowing the main (usually the UI) thread to run without being blocked/slowed down.”

- “Living standard” status. Supported on all major browsers.

Web Workers

- As you already know, shared memory is the source of multithreading complexity
- Web workers have a simplified model, but there are both pros and cons to this
- In short, there isn’t shared memory
  - Objects sent between worker and code that spawned the worker via message queues
  - Objects either copied or transferred when put in a message queue

Web Workers

- Constructor with Worker construction function
  - Takes 1 parameter for the URL for the .js file
- Send messages to the worker with the postMessage function
  - Certain objects (functions) cannot be sent
  - Other objects are deep copied
  - Other objects are transferred (i.e. moved from thread that posts object to thread that receives it, making it accessible only on receiving thread afterwards)
    - ArrayBuffer objects are transferable

Web Workers

- onmessage event fires when a message is put into the worker’s message queue
- Simple worker.js script might look like (entire file below):

```
onmessage = function(e) {
    console.log(“Message received from main script: ” + e.data);
    postMessage(“String message sent back to main script”);
}
```

Web Workers

- Code to create the worker, setup the messaging callback, and post a message (an array object in this example):

```
var myWorker = new Worker("./mergesort_worker.js");
myWorker.onmessage = function(msg) {…};
myWorker.postMessage(sortThisArray);
```

Web Workers

- Web workers offer parallel execution, but need to think about how to use message queues to transfer data and synchronize events
- Question to answer:
  - If we can’t pass a callback function to a web worker through the message queue, then how do we generate functions that execute asynchronously, use web workers for background computation, and then invoke the callback function when complete?
- In other words, how do we implement:

function AsyncSort(array, completionCallBackFunc) {…} // (discuss)