U. Srividya
21261A05J9

FSD Assigment-1

1. Compare Event Callbacks and Threaded model.

* The Node.js event model does things differently. Instead of executing all work for each request on individual threads, work is added to an event queue and then picked up by a single thread running an event loop. the event loop grabs the top item in the event queue, executes it, and then grabs the next item. When executing code that is no longer live or has blocking I/O, instead of calling the function directly, the function is added to the event queue along with a callback that is executed after the function completes. When all events on the Node.js event queue have been executed, the Node application terminates.

* The below figure illustrates the way Node.js handle the GetFile and GetData requests.

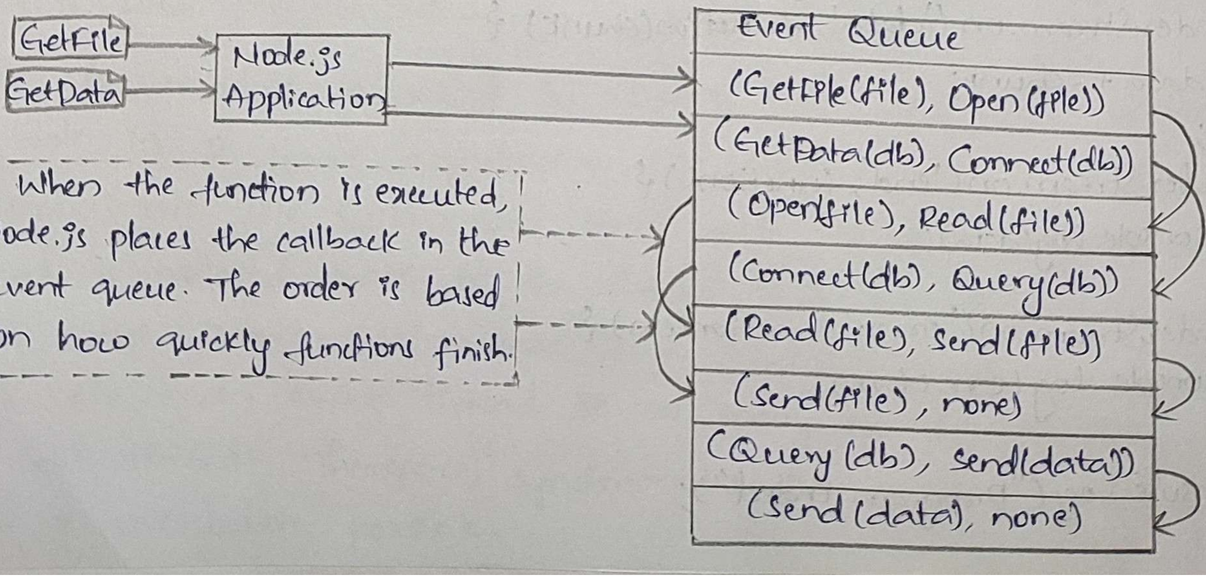
* The GetFile and GetData requests are added to the event queue. Node.js first picks up the GetFile request, executes it, and then completes by adding the open() callback function to the event queue.

* Next, it picks up the GetData request, executes it, and completes by adding the connect() callback function to the event queue.

* this Continues until there are no callback functions to be executed.

* The figure below that the events for each thread do not necessarily follow a direct interleaved order.

* For example, the Connect request takes longer to complete than the Read request, so send(file) is called before Query(db)


GetFile
GetData
Node.js Application
Event Queue
(GetFile(file), Open(file))
(GetData(db), Connect(db))
(Open(file), Read(file))
(Connect(db), Query(db))
(Read(file), send(file))
(Send(file), none)
(Query(db), send(data))
(Send(data), none)
When the function is executed, Node.js places the callback in the event queue. The order is based on how quickly functions finish.

2. What are streams in Nodejs? Describe the different types of streams with examples.

Streams are one of the fundamental concepts that power Node.js applications. They are data-handling method and are used to read or write input into output sequentially.
Streams are a way to handle reading/writing files, network communications, or any kind of end-to-end information exchange in an efficient way.
There are 4 types of streams in Node.js:-
1. Writable: streams to which we can write data. For example, fs.createWriteStream() let us write data to a file using streams.
2. Readable:- streams from which data can be read. For example: fs.CreateReadStream() let us read the contents of a file.
3. Duplex:- Streams that are both Readable and Writable. For example: net.Socket'
4. Transform:- streams that can modify or transform the data as it is written and read. For example, in the instance of file-compression you can write compressed data and read decompressed data to and from a file.

```
//read stream example:
var fs=require("fs");
var data='';
var readerStream= fs.createReadStream('file1.txt');
readerStream.setEncoding("UTF8");
readerStream.on('data', function(chunk) {
   data+=chunk;
});
readerStream.on('end', function(){
  console.log(data);
});
readerStream.on('error',function(err){
  console.log(err.stack);
});
console.log("Program Ended");
```

```
//write stream example:-
Const fs=require("fs");
Const dataStream= fs.createWriteStream('data.txt');
dataStream.write("First line of data\n');
dataStream.write('second line of data.\n');
dataStream.end(()=>{
   console.log('Finished writing data to file.');
})
//example for duplex:-
Const {Duplex} =require('stream');
Const inoutStream=new Duplex({
  write(chunk, encoding ,callback){
   console.log(chunk.toString());
   callback();
 },
 read(size){
  this.push(String.fromCharCode(this.currentCharCode++));
   if (this.currentCharCode>90){
     this.push(null);
   }
  }
});
inputStream.currentCharCode=65;
Process.stdin.pipe(inoutStream).pipe(process.stdout);
```

3. Difference between synchronous and asynchronous file system calls in Node.js. What are the advantages and disadvantages of each?

* The fs module provided in Node.js makes almost all functionality available in 2 forms: asynchronous and synchronous.
* synchronous file system calls block until the call completes and then control is related back to the thread. This has advantages but can also cause severe performance issue in Node.js if synchronous calls block the main event thread ~~or~~ too many of the background thread -d pool threads. Therefore, synchronous file system calls should be limi -ted in use when possible.

* Asynchronous called are placed on the event queue to be run later. This allows the calls to fit into the Node.js event model; however, this can be tricky when executing your code because the calling thread continues to run before the asynchronous call gets picked up by the event loop. *

* Asynchronous calls require a callback function as an extra parameter. The callback function is executed when the file system request completes, and typically error as its first parameter.

Synchronous file system

Advantages:-

1. simplicity:- The synchronous operations are straightforward and easier to understand.

2. Blocking operations:- Useful for scripts or scenarios where subsequent code relies on the completion of file operations, such as during initialization tasks.

Disadvantages:-

1. Blocking the Event loop:- Synchronous Operation block the event loop, Preventing other operations from completing executing untils the current operation completes.

2. poor scalability:- synchronous file operations are not suitable for applications that require high scalability or responsiveness, as they can significantly slow down the application when dealing with large files or multiple operations.

Asynchronous file system:-

Advantages:-

1. Non-blocking
2. Scalability
3. Better Resource Utilization

Disadvantages:-

1. Complexity
2. Error Handling
3. Debugging Difficulties

4. Describe the process of creating, exporting and importing a custom module in Node.js.

1. Creating a Custom Module:-

To create a custom module, you define the functionality (like functions, objects, or classes) that you want to make available in other parts of your application.

Ex:- Creating a mathOperations.js Module.

```
//mathOperations.js
function add(a,b) {
  return a+b;
}
function subtract(a,b) {
  return a-b;
}
module.exports = {
    add,
    subtract
};
```

2. Exporting a custom Module:-

Node.js use module.exports to export values (Objects, functions, or variables) from a file. Anything assigned to module.exports is made available to be imported into other files.

Exporting a single value: You can export a single function or object directly.

```
module.exports = function() {
    console.log("This is a single exported function");
};
```

Exporting Multiple values: You can export multiple function or variables by wrapping them in an object, as seen in the mathOperations.js example above.

3. Importing a Custom Module:-

Once a module has been exported, you can import it into other files using require().

Ex:- Importing the mathOperations.js Module.

```
const math = require('./mathOperations');
console.log(math.add(5,3));
console.log(math.subtract(10,4));
```

5. Explain the following modules a) dns b) crypto

a) dns :-

* If your Node.js application needs to resolve DNS domain names, look up domains, or do reverse lookups, then the dns module is helpful.

* A DNS lookup contacts the domain name server and requests records about a specific domain name.

* A reverse lookup contacts the domain name server and requests the DNS name associated with an IP address.

* The dns module provides functionality for most of the lookups that you may need to perform.

b) Crypto :-

* As the name suggests, it creates cryptographic information, or in other words, creates secure communications using secret code.

* To use crypto, you must make sure that it is loaded into your Node project.

* The easiest way to ensure crypto is loaded is to use a simple try catch (err),

for example:

```
let crypto;
try {
  crypto = require('crypto');
} catch(err) {
  Console.log('crypto support is disabled!');
}
```