File System

File System

- Node.js includes File System ‘fs’ module to access physical file system.
- The fs module is responsible for all the <u>asynchronous</u> and <u>synchronous</u> file I/O operations.
- To include the File System module, use the require() method:

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

- Common uses of the File System module:
  - → Read files
  - → Write files
  - → Update files
  - → Append files
  - → Rename files
  - → Delete files

Important methods

Method	Description
fs.readFile(fileName [,options], callback)	Reads existing file.
fs.writeFile(fileName, data[, options], callback)	Writes to the file. If file exists then overwrites the content, otherwise creates new file.
fs.open(path, flags[, mode], callback)	Opens file for reading or writing.
fs.read(fd, buffer, offset, length, position, callback)	Read a file using file descriptor
fs.write(fd, buffer, offset, length, position, callback)	Write to a file using file descriptor
fs.appendFile(file, data[, options], callback)	Appends new content to the existing file.
fs.rename(oldPath, newPath, callback)	Renames an existing file.
fs.unlink(path, callback);	Delete a file.
fs.ftruncate(fd, len, callback)	Truncates an open file.

Self-Study

Method	Description
fs.mkdir(path[, mode], callback)	Creates a new directory.
fs.readdir(path, callback)	Reads the content of the specified directory.
fs.exists(path, callback)	Determines whether the specified file exists or not.
fs.rmdir(path, callback)	Removes an existing directory.
fs.access(path[, mode], callback)	Tests a user's permissions for the specified file.
fs.chown(path, uid, gid, callback)	Asynchronous change owner.
fs.stat(path, callback)	Returns fs.stat object which includes important file statistics.
fs.link(srcpath, dstpath, callback)	Links file asynchronously.
fs.symlink(destination, path[, type], callback)	Create symbolic link asynchronously.
fs.utimes(path, atime, mtime, callback)	Changes the timestamp of the file.

Flags

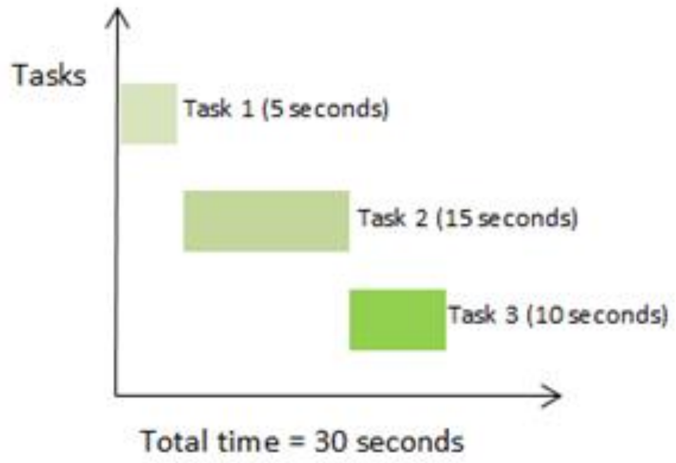
Flag	Description
r	Open file for reading. An exception occurs if the file does not exist.
r+	Open file for reading and writing. An exception occurs if the file does not exist.
rs	Open file for reading in synchronous mode.
rs+	Open file for reading and writing, telling the OS to open it synchronously. See notes for 'rs' about using this with caution.
w	Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
wx	Like 'w' but fails if path exists.
w+	Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
wx+	Like 'w+' but fails if path exists.
a	Open file for appending. The file is created if it does not exist.
ax	Like 'a' but fails if path exists.
a+	Open file for reading and appending. The file is created if it does not exist.
ax+	Like 'a+' but fails if path exists.

What is synchronous and asynchronous?

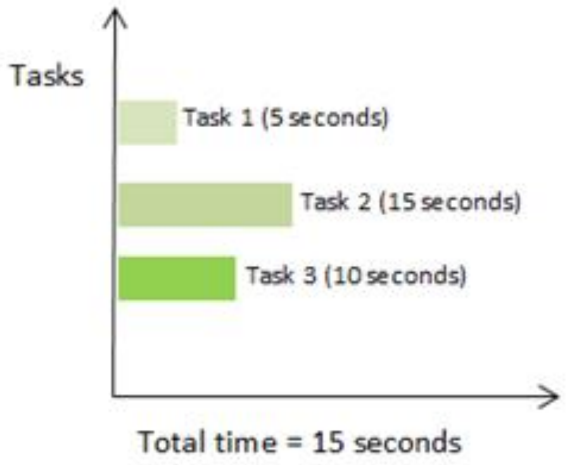
- Synchronous code is also called "blocking" as it halts the program until all the resources are available.
- Synchronous execution usually uses to code executing in sequence and the program is executed line by line, one line at a time.
- When a function is called, the program execution waits until that function returns before continuing to the next line of code.

- Asynchronous code is also known as "non-blocking". The program continues executing and doesn't wait for external resources (I/O) to be available.
- Asynchronous execution applies to execution that doesn't run in the sequence it appears in the code. The program doesn't wait for the task to complete and can move on to the next task.

What is synchronous and asynchronous?



Synchronous



Asynchronous

- Synchronous code wastes around 90% of CPU cycles waiting for the network or disk to get the data, but the Asynchronous code is much more performing.
- Using Asynchronous code is a more efficient to have concurrency without dealing with multiple execution threads.

Callbacks

- Callback is a special type of function passed as an argument to another function.
- A callback function is called when a task is completed, thus helping in preventing any kind of blocking, allowing other code to run/execute in the meantime.
- Callbacks help us make asynchronous calls.
- In Node.js, once file I/O is complete, it will call the callback function. So there is no blocking or waiting for file I/O operations.
- This makes Node.js highly scalable, as it can process high number of requests without waiting for any function to return the result.

Reading a File

- fs.readFile() method is used to read the physical file <u>asynchronously</u>.

```
fs.readFile(fileName[, options], callback);
```

<u>filename</u>: Full path and name of the file as a string.

<u>options</u>: The options parameter can be an *object* or *string* which can include encoding and flag. The default encoding is <u>null</u> and default flag is "<u>r</u>". If no encoding is specified, then the raw buffer is returned.

<u>callback</u>: A function with two parameters <u>err</u> and <u>data</u>. This will get called when read operation completes.

Reading a File

- fs.readFileSync() method is used to read the physical file synchronously.

```
fs.readFileSync(fileName[, options]);
```

filename: Full path and name of the file as a string.

options: The options parameter can be an object or string which can include encoding and flag. The default encoding is null and default flag is "r". If no encoding is specified, then the raw buffer is returned.

Writing to a File

- fs.writeFile() method is used to write data to a file asynchronously. If file already exists, then it overwrites the existing content otherwise it creates a new file and writes data into it.

```
fs.writeFile(filename, data[, options], callback)
```

filename: Full path and name of the file as a string.

data: The content to be written to a file.

options: The options parameter can be an *object* or *string* which can include encoding, mode and flag. The default encoding is 'utf8', default file mode (permissions) is octal value '0o666' and default flag is 'w'.

callback: A function with parameter err. This will get called when write operation completes.

Writing to a File

- fs.writeFileSync() method is used to write data to a file synchronously. If file already exists, then it overwrites the existing content otherwise it creates a new file and writes data into it.

```
fs.writeFileSync(filename, data[, options])
```

filename: Full path and name of the file as a string.

data: The content to be written to a file.

options: There are 3 optional parameters

→ **encoding**: It is a string that specifies the encoding of the file. The default value is 'utf8'.

→ **mode**: It is an integer that specifies the file mode. The default is octal value '0o666'.

→ **flag**: It is a string that specifies the flag used while writing to the file. The default value is 'w'.

The readline module

- The readline module in Node.js allows the reading of input stream line by line.
- This module wraps up the standard output and process standard input objects. The Readline module makes it easier to input and read the output given by the user.
- The syntax for including the Readline module in your application

```
var readline = require('readline’);

let rl = readline.createInterface(process.stdin, process.stdout);
```

- The createInterface() method takes two arguments.
  - The first argument is for the standard input
  - The second one is for reading the standard output

The readline module: important methods

1. readline.question()
2. readline.close()
3. readline.resume()
4. readline.setPrompt()
5. readline.prompt()
6. readline.on()

Rename a File

- fs.rename() method is used to asynchronously rename a file from the file system.

```
fs.rename(oldPath, newPath, callback);
```

oldPath: It holds the path of the file that has to be renamed. It can be a string, buffer or URL.

newPath: It holds the new path that the file has to be renamed. It can be a string, buffer or URL.

callback: It is the function that would be called when the method is executed. It has an optional argument for showing any error that occurs during the process.

Note: Synchronous version of this method is also available

Delete a File

- fs.unlink() method is used to asynchronously remove a file or symbolic link from the file system.
- This function does not work on directories, therefore it is recommended to use fs.rmdir() to remove a directory.

```
fs.unlink(fileName, callback);
```

filename: It is a string, Buffer or URL which represents the file or symbolic link which has to be removed.

callback: A function that would be called when the method is executed.

Note: Synchronous version of this method is also available

Append a File

- fs.appendFile() method is used to asynchronously append the given data to a file. A new file is created if it does not exist.
- The options parameter can be used to modify the behavior of the operation.

```
fs.appendFile(path, data[, options], callback)
```

path: It is a String, Buffer, URL or number that denotes the source filename or file descriptor that will be appended to.

data: It is a String or Buffer that denotes the data that has to be appended.

options: It is an string or an object that can be used to specify optional parameters that will affect the output. It has three optional parameters:

- **encoding**: It is a string which specifies the encoding of the file. The default value is 'utf8'.
- **mode**: It is an integer which specifies the file mode. The default value is '0o666'.
- **flag**: It is a string which specifies the flag used while appending to the file. The default value is 'a'.

callback: It is a function that would be called when the method is executed.

- **err**: It is an error that would be thrown if the method fails

Append a File

- fs.appendFileSync() method is used to synchronously append the given data to a file. A new file is created if it does not exist.
- The options parameter can be used to modify the behavior of the operation.

```
fs.appendFileSync(path, data[, options])
```

path: It is a String, Buffer, URL or number that denotes the source filename or file descriptor that will be appended.

data: It is a String or Buffer that denotes the data that has to be appended.

options: It is an string or an object that can be used to specify optional parameters that will affect the output. It has three optional parameters:

- **encoding**: It is a string which specifies the encoding of the file. The default value is 'utf8'.
- **mode**: It is an integer which specifies the file mode. The default value is '0o666'.
- **flag**: It is a string which specifies the flag used while appending to the file. The default value is 'a'.

Open a File

- You can open a file for reading or writing using the fs.open() method.

```
fs.open(fileName, flags[, mode], callback);
```

filename: Full path and name of the file as a string.

flags: Flags to perform operation.

mode: The mode can be read, write or read/write. The default is read/write.

callback: A function with two parameters err and fd. This will get called when file open operation completes.

Reading a File

fs.read() method is used to read a file using file descriptor.

```
fs.read(fd, buffer, offset, length, position, callback)
```

fd: File descriptor returned by fs.open() method.

buffer: Stores the data fetched from the file.

offset: The position in buffer to write the data to.

length: An integer that specifies the number of bytes to read.

position: Specifies where to begin reading from in the file. If position is null or -1 , data will be read from the current file position, and the file position will be updated.

callback: The callback function accepts the three arguments - err, bytesRead, buffer.

Writing to a File

fs.write() method is used to write buffer to a file using file descriptor.

The fs.write() method can also be used without a buffer by simply using a string variable.

```
fs.write(fd, buffer, offset, length, position, callback)
```

```
fs.write(fd, string, position, encoding, callback)
```

fd: File descriptor returned by fs.open() method.

buffer: Data to be written to a file

offset: Determines the part of the buffer to be written.

length: An integer specifying the number of bytes to write.

position: Position from the beginning of the file where this data should be written.

callback: The callback function accepts the three arguments - err, bytesWritten, buffer

Truncate a File

- fs.ftruncate() method is used to truncate an open file from the file system.

```
fs.ftruncate(fd, length, callback);
```

fd: This is the file descriptor returned by fs.open().

length: This is the length of the file after which the file will be truncated.

callback: It is the function that would be called when the method is executed. It has an optional argument for showing any error that occurs during the process.

Practice

Write Node.js application(s) –

1) To accept input & file name from the user from command line and write it to a file. Then read the contents of this file and display on the console.

2) To count the number of words in a file and display on the console.

3) To count the number of lines in a file and display the same on the console.

4) To count the number of vowels in a file and display the same on the console.

5) To search a particular word in file and display the count on the console. Accept the word from the user, as a command line argument.

Note: a) Do these programs using synchronous & asynchronous methods.
b) Display the output on a web page using Node.js web server.

Practice

Write Node.js application(s) –

1) To display an image on a web page.

2) To create a simple web server using Node.js that displays the college information on a web page.

3) *Accept* file name from the user using readline and display
   a) alternate characters from a file
   b) alternate words from a file
   c) alternate lines from a file

4) *Accept* file name from the user using readline and display contents of a file in the reverse order
   a) on console
   b) on a web page

<u>Note</u>: Do these programs using synchronous & asynchronous methods.

Practice

Write Node.js application(s) –

5) To accept two file names and the contents from the user and write to two files. Then append contents of a first file to a second file. Then display the contents of a second file on the console.

6) To read contents from two files and merge the contents into a third file by converting it into uppercase.

Note: Do these programs using synchronous & asynchronous methods.