Node.js FS Overview

Lecture 06: Node.js File System (FS) – Access using Files & Streams

Course: Advanced Web Technologies (CSC337)
Time: 2 Hours
Lecture Type: Theory

Lecture Outline

1. **Introduction to Node.js File System (FS)**
2. **Understanding Files and Streams in Node.js**
3. **Synchronous vs Asynchronous File Handling**
4. **Reading and Writing Files**
5. **Working with Streams (Readable, Writable, Duplex, and Transform)**
6. **Practical Use Cases**
7. **Student Activity (Quiz & Discussion)**

---

1. Introduction to Node.js File System (FS)

- Node.js provides a built-in module called **File System (FS)** that allows interaction with the file system.
- The **fs** module can be used for:
  - Reading files
  - Writing files
  - Deleting files
  - Renaming files
  - Working with directories

---

2. Understanding Files and Streams in Node.js

Files

- Files are discrete units of data stored on a disk.
- Node.js allows us to **read**, **write**, and **modify** files in various formats like **`` `.txt` ``**, **`` `.json` ``**, **`` `.csv` ``**, etc.

Streams

- Streams handle data in chunks rather than loading it all into memory.
- Useful for handling **large files** efficiently.
- Four types of streams:
    1. **Readable Streams** – For reading data (e.g., reading a file)
    2. **Writable Streams** – For writing data (e.g., writing to a log file)
    3. **Duplex Streams** – For both reading and writing (e.g., sockets)
    4. **Transform Streams** – For modifying data (e.g., compressing a file)

---

3. Synchronous vs Asynchronous File Handling

Synchronous (Blocking)

- Code execution waits until the file operation is completed.
- Example:

    ```
    javascript

    const fs = require('fs');
    let data = fs.readFileSync('file.txt', 'utf8');
    console.log(data);
    ```

- **Problem:** Slows down the application if the file is large.

Asynchronous (Non-Blocking)

- Code execution continues while file operations run in the background.
- Example:

    ```
    javascript

    const fs = require('fs');
    fs.readFile('file.txt', 'utf8', (err, data) => {
        if (err) throw err;
        console.log(data);
    });
    ```

- **Advantage:** Application remains responsive.

---

4. Reading and Writing Files

Reading a File (Async)

```
javascript

const fs = require('fs');
fs.readFile('example.txt', 'utf8', (err, data) => {
    if (err) {
        console.error("Error reading file:", err);
    } else {
        console.log("File contents:", data);
```

```
    }
});
```

Writing to a File (Async)

```
javascript

fs.writeFile('output.txt', 'Hello, Node.js!', (err) => {
    if (err) throw err;
    console.log('File has been saved!');
});
```

---

5. Working with Streams

Readable Stream

```
javascript

const fs = require('fs');
const readableStream = fs.createReadStream('largeFile.txt', 'utf8');

readableStream.on('data', (chunk) => {
    console.log("Received chunk:", chunk);
});
```

Writable Stream

```
javascript

const writableStream = fs.createWriteStream('output.txt');
writableStream.write('Writing some data...\n');
writableStream.end();
```

Piping Streams

- Pipes allow direct transfer between readable and writable streams.
- Example:

  ```
  javascript

  const fs = require('fs');
  const readStream = fs.createReadStream('input.txt');
  const writeStream = fs.createWriteStream('output.txt');

  readStream.pipe(writeStream);
  ```

- **Benefit:** No need to store data in memory before writing.

---

6. Practical Use Cases

- **Logging System:** Writing logs into a file using streams.
- **File Uploads:** Handling large file uploads efficiently.

- **Real-time Data Processing:** Processing large datasets in chunks.

---

7. Student Activity

Quiz (10 Minutes)

1. What is the main difference between synchronous and asynchronous file operations in Node.js?
2. Name the four types of streams in Node.js.
3. What is the benefit of using streams over regular file reading?
4. Write a short code snippet to read a file asynchronously.
5. How does the ``pipe()`` function work in Node.js streams?

Discussion (10 Minutes)

- Discuss real-world examples where file streaming is useful (e.g., video streaming, chat applications).
- Debate the advantages of using asynchronous file handling over synchronous methods.

Conclusion

- **The FS module** in Node.js is crucial for handling files and directories.
- **Streams** provide an efficient way to process large files.
- **Asynchronous operations** enhance application performance by avoiding blocking operations.

Next Lecture: MongoDB – Database Creation & Access