Advanced Node.js Interview Questions & Answers (with Code Examples)

1. Explain the Event Loop in Node.js.

The event loop handles asynchronous callbacks. It allows Node.js to perform non-blocking I/O operations by offloading operations to the system kernel whenever possible.

2. What is libuv and why is it important?

libuv is a C library that implements the event loop, thread pool, and async I/O operations. It makes Node.js cross-platform and highly efficient.

3. How do you handle child processes in Node.js?

You can spawn child processes using the child_process module.

```
const { fork } = require('child_process');
const child = fork('./child.js');

child.on('message', msg => console.log('Message from child:', msg));
child.send({ hello: 'world' });

// child.js
process.on('message', msg => {
  console.log('Message from parent:', msg);
  process.send({ reply: 'Hi parent!' });
});
```

4. What are Worker Threads?

Worker threads allow CPU-heavy JavaScript operations to run in parallel, separate from the main event loop.

```
const { Worker, isMainThread, workerData } = require('worker_threads');

if (isMainThread) {
  const worker = new Worker(__filename, { workerData: 10 });
  worker.on('message', result => console.log('Fibonacci:', result));
} else {
  function fib(n) { return n < 2 ? n : fib(n-1) + fib(n-2); }
  parentPort.postMessage(fib(workerData));
}
```

5. Explain clustering in Node.js.

Clustering enables running multiple Node.js processes to utilize all CPU cores.

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

if (cluster.isPrimary) {
  for (let i = 0; i < numCPUs; i++) cluster.fork();
  cluster.on('exit', worker => console.log(`Worker ${worker.process.pid} died`));
} else {
  http.createServer((req, res) => res.end('Hello from Worker ' + process.pid)).listen(3000);
}
```

6. What is the EventEmitter class used for?

The EventEmitter allows you to create and listen for custom events.

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}

const emitter = new MyEmitter();
emitter.on('start', (msg) => console.log('Started:', msg));
emitter.emit('start', 'Event triggered');
```

7. How do you handle errors globally?

Handle uncaught exceptions and unhandled promise rejections using process events.

```
process.on('uncaughtException', (err) => {
  console.error('Uncaught Exception:', err);
});

process.on('unhandledRejection', (reason, promise) => {
  console.error('Unhandled Rejection:', reason);
});
```

8. Explain async/await with an example.

Async/await simplifies asynchronous code built on promises.

```
function asyncTask() {
  return new Promise(resolve => setTimeout(() => resolve('Task Done!'), 1000));
}

async function run() {
  console.log(await asyncTask());
}

run();
```

9. How to optimize Node.js performance?

• Use asynchronous APIs instead of synchronous ones.

• Use clustering or worker threads for scalability.

• Implement caching (Redis or memory cache).

• Use PM2 for process management.

• Profile performance using --inspect or Clinic.js.

10. How to secure a Node.js application?

• Use Helmet.js for HTTP header protection.

• Avoid eval() and user-supplied code execution.

• Validate and sanitize user inputs.

• Store secrets in environment variables (.env).

• Use HTTPS and proper session management.

End of Advanced Node.js Interview Questions — Prepared for developers aiming for senior roles.