Ø=ÜØ Day 23 – Child Process in Node.js

Ø=Ý6 What is a Child Process?
Node.js ka Child Process Module allow karta hai ki tum ek new system process bana sako —
taaki multiple tasks parallel me perform kiye ja sakein without blocking the main thread.
Ø=ÜÌ Node.js is single-threaded, lekin child_process ka use karke tum multiple processes create kar sakte ho for background jobs.

' Real-Life Analogy:
Tumhara main program ek manager hai, aur child process ek worker jise manager bolta hai:
"Ja jaake file compress kar aa" ya "external script chala ke result de aa".

Ø=Ý' Common Methods in child_process Module:
Method”FW67&— F–öà
exec()”W‡FW&æ Â 6öÖÖ æB 'Vâ ¶ 'F † ’ ‡&WGW&ç2 ÷WG WB 2 7G&–ær•
execFile()•7 V6–`ic file execute karta hai (faster than exec)
spawn()•7G&V ×2 °e through large data process karta hai
fork()”æöFRæ§2 67&— B 'Vâ ¶ 'F † ’ v—F, • 2 ‡W6VB f÷" 6ÇW7FW&–ær•

' Install/Import Module
js
CopyEdit
```
const { exec, spawn, fork } = require("child_process");
```

Ø=Ý8 1. exec() – Run shell command & get output (buffer-based)
js
CopyEdit
```
const { exec } = require("child_process");

exec("node -v", (err, stdout, stderr) => {
  if (err) {
    console.error("Error:", err);
  } else {
    console.log("Node version:", stdout);
  }
});
```
Ø>Ýà exec output ko buffer me return karta hai (not suitable for large data).

Ø=Ý8 2. spawn() – For large data (stream-based)
js
CopyEdit
```
const { spawn } = require("child_process");

const process = spawn("ping", ["google.com"]);

process.stdout.on("data", (data) => {
  console.log(`Output: ${data}`);
});
```
Ø>Ýà spawn() output ko stream me deta hai (efficient for big data).

Ø=Ý8 3. fork() – Run another Node.js script with communication

js

CopyEdit
```
// parent.js
const { fork } = require("child_process");
const child = fork("child.js");

child.send("Hello from parent");

child.on("message", (msg) => {
  console.log("Message from child:", msg);
});
```
js

CopyEdit
```
// child.js
process.on("message", (msg) => {
  console.log("Child received:", msg);
  process.send("Hello from child");
});
```
Ø>Ýà fork() is best for IPC (Inter Process Communication) between two Node.js scripts.

Ø=ÜÈ Why Use Child Processes?

Use Case•v€y?

Image / Video processing”†V vy CPU task in child thread

File Compression / Decompression”Æöær F 6² r void blocking main thread

Real-time monitoring (logs, memory)• arallel process helps

Running Python / Shell scripts”W‡FW&æ Â FööÇ2 –çFPgration

Clustering HTTP server”† æFÆR Ö÷&R G& `fic efficiently

Ø>Ýà Backend Example:

js

CopyEdit
```
const { exec } = require("child_process");

app.get("/backup", (req, res) => {
  exec("mongodump --db=todoApp", (err, stdout, stderr) => {
    if (err) return res.send("Backup failed");
    res.send("Backup successful");
  });
});
```
Use-case: DB backup via system command

Ø>Ýà Summary Table:

Method•W6R 6 6YReturns

exec()•6†VÆÂ 6öÖÖ æG2 ‡6Ö ÆÂ F F ™Buffer

spawn()”Æöær×'Vææ–ær &ö6W76W9Stream
fork()”æöFR×FòÔæöFR &ö6W79Object + IPC
execFile()•'Vâ   7 V6–`ic executable”'V`fer

Ø=Ý Conceptual Diagram:
scss
CopyEdit

```
Main Node.js Process
     %
 % % % % %4% % % % %
 %  Child 1 (exec !’ command)
 %  Child 2 (spawn !’ stream)
 %  Child 3 (fork !’ node script)
 % % % % % % % % % %
```

' Bonus Interview Tip:
Ø=Ü¬ Q: How is spawn() different from exec()?
A: exec() returns output in buffer (good for small data), while spawn() gives streaming output — better for large output/data.