Create child process
Four methods `spawn` `fork` `execFile` `exec`

1. Which of the following about `spawnSync()` and `spawn()` are correct?

a) `spawnSync()` will create a synchronize child process.
b) A synchronize child process will block the event loop until the process is finished (exited) or terminated.
c) All the methods in child_process module are used to build asynchronous process.
d)`spawnSync()` will not return until the child process is closed.
e) For both methods, if the shell option is enabled, passing a user input directly to the method may cause the unexpected command execution.
A, b, d,e

5. Which of the following statement about method `execFile()` is not correct?

a) `execFile` can be used to execute a executable program in a child process.
b) `execFile` has a maximum buffer which limits the largest amount of data allowed in stdout or stderr, if not specified it's 200*1024.
c) Sometimes you will get an error "maxBuffer exceeded" because the return data of the child process exceeds the amount of maxBuffer.
d)`execFile` itself will return data of string or buffer type.
d
The data in the output is returned by the child process.

2. What is the output of the following code

```
console.log("hello world");
const execFile = require( 'child_process').execFile;
const child = execFile( 'node', [ '--version'], ( error, stdout, stderr) => {
   if ( error) {
      console.error( 'stderr', stderr);
      throw error;
   }
   console.log( 'stdout', stdout);
});
console.log("miao");
```

```
0587367681:LearnNodejs mac$ node firstFile.js
hello world
miao
stdout v8.12.0
```

4. Which of the following about `exec()` is wrong?

a) `exec` is used to run a command.
b) It doesn't have a maxBuffer parameter because the default data return from the callback function is string type.
c) The error parameter of callback function will be null if the command is executed correctly.
d)`exec()` will spawn a shell first and then run the command in the newly-created shell.

B

6. **What are correct statements about the code:** ```const exec = require('child_process').exec;
exec('cat *.js | wc -l', (error, stdout, stderr) => {
  if (error) {
    console.error(`exec error: ${error}`);
    return;
  }
  console.log(`stdout: ${stdout}`);
  console.log(`stderr: ${stderr}`);
});```
(assume only one .js file is accessible)

```
0587367681:LearnNodejs mac$ node firstFile.js
stdout:       132

stderr:
```

2. Which of the following statement about `fork` is true?
A The first parameter modulePath is a path to the module to run in the child process.
B The child process created by `fork` will have a communication channel which allows messages passing between parent and child process.
C If the silent optional parameter is set to be true, the output will pipe to child.stdout.
D You can use subprocess.send() to send message to the child process.
A b c d

3.**The following statements are about message sending and receiving between parent and child process. Which is wrong?**
https://nodejs.org/dist/latest-v6.x/docs/api/child_process.html#child_process_subprocess_send_message_sendhandle_options_callback

A `subprocess.send()` method can be used to send messages to the child process.
B `subprocess.send()` will return false only if the communication is closed
C Child process can receive the messages from parent process by `process.on(`message`)`
D `process.on(`message`)` emits an event “message” when message comes in.

1. **What is the expected output of the following code:** parentProcess.js: ```var childProcess = require('child_process');
var n = childProcess.fork('./childProcess.js');
n.on('message', function(m) {

  console.log('Parent Process Listen: ', m);
});
n.send({ hello: 'From Parent Process' });```
childProcess.js: ```process.on('message', function(m) {

```
  console.log('Child Process Listen:', m);
});process.send({ hello: 'From Child Process' });```
```

```
0587367681:LearnNodejs mac$ node parentprocess.js
Child Process Listen: { hello: 'From Parent Process' }
Parent Process Listen:  { Hello: 'From Child Process' }
```

The expected output is “Child Process Listen: { hello: 'From Parent Process' }
Parent Process Listen: { Hello: 'From Child Process' }”
The expected output is “Parent Process Listen: { hello: 'From Child Process' }
Child Process Listen: { Hello: 'From Parent Process' }”
The expected output is “Child Process Listen: { hello: 'From Child Process' }
Parent Process Listen: { Hello: 'From Parent Process' }”

9**. What are the right statements about using different method in creating a child process?**

A When we want to manipulate a big amount of data returned by child process, `execFile()` is a good choice.
B `spawn()` is often used when you want to deal with the data coming from the child process.
C `execFile()` is built upon `spawn()`, while `exec()` is built upon `execFile()`.
D `exec()` will return all the data in buffer at one time when the child process is finished or terminated.
E `fork()` will build up a pass for messages between parent process and child process.
bcde

10**. what are the correct statements about `spawn()`**

A `spawn()` will return a stream when the child process is executed.
B When you use spawn to create a child process to convert a picture, the user will see the picture even before the picture is finish converting.
C Since `spawn()` is based on stream, the output data can exceed the V8 heap limit which is 1GB on 64-bit system.
D Using `spawn()` will take up less internal storage compared to `exec()`
Abcd

Framework

3. **Which statement is wrong about Node.js framework?**

a) Node.js is based on a couple of libraries like V8, libuv, zlib and so on.
b) The libraries are on the first layer above which are Node.js bindings, then the APIs we actually use.
c) All the layers are written in js.
d) Node.js bindings are used to expose the C/C++ APIs to JS environment.
c

Event

7. **In terms of how to use EventEmitter, what are the correct statements?**

a) We can make a new instance of EventEmitter class and the methods can be called by `newInstance.emit(...)`.

b) We can make our own class inheriting EventEmitter class

c) It is not allowed to register the event of the same name to the same emitter.

A b

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


var em = new events.EventEmitter();
// var em2 = new events.EventEmitter();


em.on('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

em.on('FirstEvent', function (data) {
    console.log('second subscriber: ' + data);
});
```

```
0587367681:LearnNodejs mac$ node firstFile.js
First subscriber: This is my first Node.js event emitter example.
second subscriber: This is my first Node.js event emitter example.
0587367681:LearnNodejs mac$
```

8. **What are correct statements about the code:** ```var events = require('events');
var emitter= new events.EventEmitter();
emitter.on('hi',function(one) { console.log('I'm', one);
})
emitter.on('hi',function(one) { console.log('He is', one);
})
emitter.emit('hi', 'Sam');```

9. **Which of the following is wrong about `newListener` event?**

A. It will be emitted before a listener is added to the listener array.

B. If the listener of the same name is also registered in the callback function of `newListener`, it will be inserted before the listener that added in the normal process.

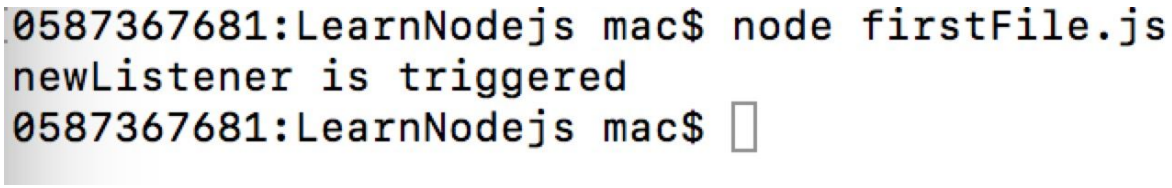
C. The callback function of `newListener` will be triggered every time a new listener is registered.
D. If multiple listeners of the same name are registered, The callback function of `newListener` will only be triggered once.

D

8.**What is the expected output of the following code?**

```const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();
myEmitter.on('newListener', () => {   console.log("newListener is triggered");
 });
myEmitter.on('removeListener', () => {   console.log('Removed');
});```

Nothing
“Removed”
“newListener is triggered”
“newListener is triggered Removed”
c




10**. What is the expected output of the code?**

```const EventEmitter = require('events');
const myEmitter = new MyEmitter();
myEmitter.once('newListener', (event, listener) => {
 if (event === 'event') {
   myEmitter.on('event', () => {
     console.log('B');
   });
});
myEmitter.on('event', () => {
 console.log('A');
});
myEmitter.emit('event');```

A. B A
B. A
C. A B
D. B

A correct

# Process

**4. What are some of the properties of a Process?**

Stdin
Stdout
Stderr
Pid
Ppid
readableLength
writableLength
abcde

**5. Which is wrong about `process.pid`?**

A process.pid will return the PID of the process
B process.ppid will return the PID of the parent process that creates the current process
C The process.pid can be changed by writing node.js codes.
D The PID is assigned to each process by operating system and every process has a unique PID.
c

**6. In terms of using `exit()` method to exit a process, what are the correct statements?**

A It's a good practice to use `exit` whenever you want to exit the process.
B In most situations, you don't need to call `exit()` explicitly since Node.js will take care of exiting the process when it is finished.
C If you want to exit when the process has an error, it's better to use `process.exitCode = 1` than `process.exit(1)`.
D A even better solution is using error condition like: `if (error) {Deal with error and return;}`
B c d

1.Which of the following is wrong about `error` event?

A. If there is no listener registered for the `error` event, the error is thrown when it occurs.
B. The Node.js process will crash after the error is thrown. (stack trace)
C. To prevent crashing, register at least one listener for `error` event to deal with errors.
D. Only the process itself can emit the `error` event, we can not use code to emit an event directly.

2. What is the expected output of the following code dealing with error?

```
```const myEmitter = new MyEmitter();
myEmitter.on('error', (err) => {
```
```

```
  console.error('whoops! there was an error');
});
myEmitter.emit('error', new Error('whoops!'));```
```

```
0587367681:LearnNodejs mac$ node firstFile.js
whoops! there was an error
0587367681:LearnNodejs mac$
```

```
0587367681:LearnNodejs mac$ node firstFile.js
events.js:183
      throw er; // Unhandled 'error' event
      ^

Error: whoops!
    at Object.<anonymous> (/Users/mac/Documents/FrontEnd/LearnNodejs/firstFile.j
s:114:25)
    at Module._compile (module.js:653:30)
    at Object.Module._extensions..js (module.js:664:10)
    at Module.load (module.js:566:32)
    at tryModuleLoad (module.js:506:12)
    at Function.Module._load (module.js:498:3)
    at Function.Module.runMain (module.js:694:10)
    at startup (bootstrap_node.js:204:16)
    at bootstrap_node.js:625:3
```

A. Nothing will be printed.
B. “Whoops! There was an error”
C. Print out the stack trace.

https://flaviocopes.com/nodejs-streams/

3.Why we use stream in Node.js?

A. We can use `readFile` and `writeFile` to transfer data, but it will treat the file as a whole. If the file is big, it will take too much internal storage.
B. Stream will split the data going to be transferred and transfer it as several blocks.
C. One advantage of using stream is that you don’t need to load large amounts of data to internal storage.
D. The other advantage is it will take less time to start processing data because you can start processing when some of the data is available.

(all correct)

4.Which of the following statement about `pipe` is wrong?

A. Write operation will be much slower than read, it may cause data losing. In order to fix it, pipe is created.

B. Pipe can read data from readable stream and write to writable stream, and also from writable stream to readable stream.
C. Pipe can deal with errors happened in data transferring.
D. Pipe can deal with informing the writable stream there is no more data from readable stream.

5. What are the true statements about the modes of readable stream?
A. There are two modes of a readable stream, flowing and paused.
B. In flowing mode, when the data is available, it will emit an event.
C. In paused mode, `stream.read()` has to be called to read data.
D. In flowing mode, `stream.read()` has to be called to read data.

https://nodejs.org/es/docs/guides/event-loop-timers-and-nexttick/
6. Which of the following statement about event loop is wrong?
A. Node.js will initialize the event loop when it starts.
B. There are 6 phases of event loop: timers, pending callbacks, idle and prepare, poll, check and close callbacks.
C. All the phases share a FIFO queue.
D. When the queue is exhausted or the callback limit is reached, the event loop will move to the next phase.

7.What are correct statements about timers?
A. Timer specifies an accurate threshold after which a callback will be executed.
B. Timers phase executes callbacks scheduled by setTimeout() and setInterval().
C. Operating system scheduling or running other callbacks may delay the actual time to execute the callback.
D. The poll phase controls when the timers are executed.