nodeJS

2021-08-13
Hong Hyo Sang

Content

I. Nods.js

1. NodeJS ?
2. Non Blocking & Thread
3. Module
4. 내장 객체
5. 내장 모듈

참고 : https://nodejs.org/ko/docs/

1. NodeJS

01. NodeJS ?

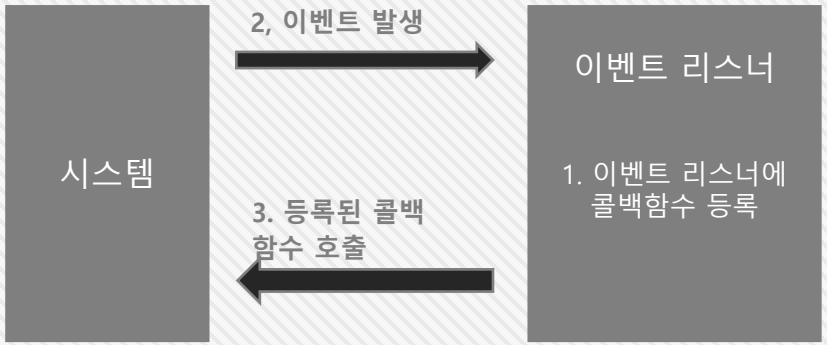
- Chrome V8 Javascript엔진으로 빌드 된 javascript 런타임

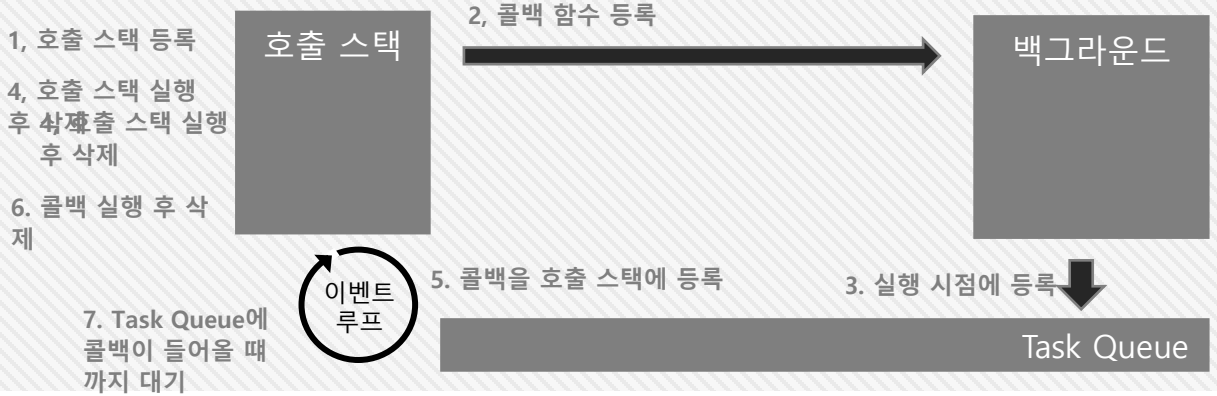
내부구조

- V8 : 오픈 소스 자바스크립트 엔진
- libuv : 이밴트 기반, 논 블로킹 I/O 모델을 구현

이벤트 기반

- 이벤트가 발생할 때 미리 지정해둔 작업을 수행 하는 방식
- Event Listener에 Callback 함수를 등록 하여 수행 하는 것을 의미함
- Event Loop : 이벤트가 동시에 발생 하였을 때 어떤 순서로 콜백 함수를 어떤 순서로 호출 할지 판단 즉 이벤트 발생시 콜백 함수를 관리
- Background : 이벤트 리스너들이 대기 하는 곳
- Task Queue : 콜백 큐라고 불리우며 이벤트 호출 시 백그라운드에서는 task queue로 이벤트 리스너의 콜백 함수를 보냄



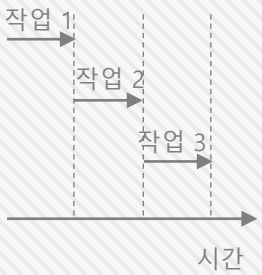


2. Non Blocking

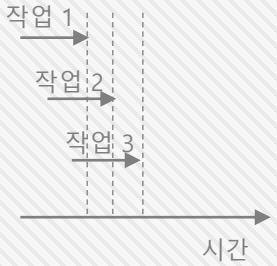
01. Non Blocking & Blocking

- Non Blocking : 작업이 완료될 때 가지 대기하지 않고 다음 작업 수행
- Blocking : 이전 작업이 끝난 후 다음 작업 수행

 # Non Blocking

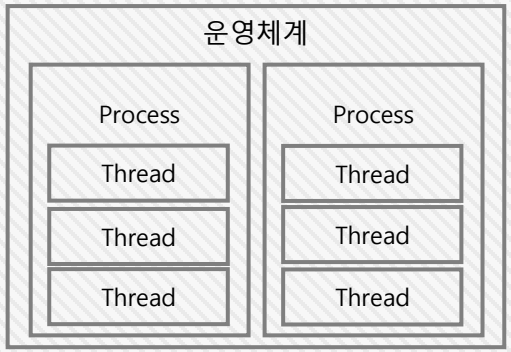


 # Blocking



02. Single Thread

- 직접 제어 할 수 있는 Thread가 하나뿐 인 것



 # Process

: 운영체계에서 할당하는 작업 단위로 메모리 등의 자원 공유 없음

 # Thread

: Process 내부에서 실행 되는 작업 단위로 메모리 등의 자원 공유 있음

03. Multi Thread

- 하나의 Process안에서 여러 개의 Thread사용
- Node12 이상에서는 Worker Thread를 사용 하여 적용됨

Multi Thread	Multi Process
하나의 Process 안에서 여러 개의 Thread 사용	여러 개의 Process 사용
CPU 작업이 많을 때 사용	I/O 요청이 많을 때 사용
프로그램 어려움	프로그램 비교적 쉬움

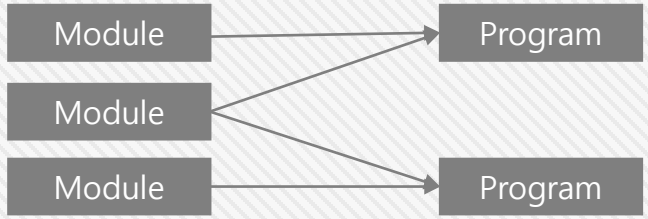
04. Node 장단점

장점	단점
Multi Thread에 비해 적은 자원 사용	CPU Core를 하나만 사용 (Single Thread)
I/O 작업이 많은 서버에 적합	CPU 작업이 많은 서버 부적합
Multi Thread 방식 보다 쉬움	Single Thread 관리 필요 (멈추지 않도록)
웹서버가 내장됨	규모가 커질수록 서버 관리 필요
자바스크립트 사용	성능이 높지 않음
JSON 형식과 쉽게 호환	

3. Module

01. Module ?

- 특정한 기능을 하는 함수나 변수들의 하나에 모아 놓은 것으로 재사용 할 수 있음
- ES2016 부터 자바스크립에 import/export 모듈 도입으로 크롬 60 부터 사용 가능




```
const sName = "홍길동";
const sAddress = "서울시 송파구 .... ";

function fPrintAddress() {
    console.log("이름 : ",  sName, "주소 :: ", sAddress);
}

function fPrintAddress01() {
    sName = "홍길자"; .
}

module.exports = {
    sName,
    fPrintAddress,
    fPrintAddress01
}
```

```
const mm = require('./module.js');

mm.fPrintAddress();  // 이름 :  홍길동 주소 ::  서울시 송파구 ....
console.log("이름 :: ", mm.sName); // 이름 ::  홍길동

mm.sName = "홍당무";
console.log("이름 :: ", mm.sName); // 이름 ::  홍당무

console.log("주소 :: ", mm.sAddress); // 주소 ::  undefined

mm.fPrintAddress01();  // TypeError: Assignment to constant variable
```

```
const {sName, fPrintAddress, sAddress, fPrintAddress01} = require('./module.js');

fPrintAddress(); // 이름 :  홍길동 주소 ::  서울시 송파구 ....
console.log("이름 :: ", sName); // 이름 ::  홍길동

sName = "홍당무"; // TypeError: Assignment to constant variable.

console.log("주소 :: ", sAddress); // undefined


fPrintAddress01(); // TypeError: Assignment to constant variable.
```

3. Module

02. ECMAScript ?

- ECMAScript을 Node.js에 사용 하기 위해서는 package.json에 모듈 선언을 하여야 함

```
{
  "type": "module“
}
```

```
export const sName = "홍길동";
const sAddress = "서울시 송파구 .... ";

export function fPrintAddress() {
    console.log("이름 : ",  sName, "주소 :: ", sAddress);
}

export function fPrintAddress01() {
    sName = "홍길자"; // TypeError: Assignment to constant variable.
}

============================================================
import {sName, fPrintAddress, fPrintAddress01} from './module.js';

fPrintAddress(); // 이름 :  홍길동 주소 ::  서울시 송파구 ....
console.log("이름 :: ", sName); // 이름 ::  홍길동
```

4. 내장 객체

01. global

- 브라우저의 window와 같은 전역 객체로 global을 생략 하고 사용 할 수 있다.

```
// module.js
export default () => global.message
;

=======================
import msg from './module.js';

global.message = "Hi";
console.log(msg()); // Hi
```

```
global: [Circular *1],
  clearInterval: [Function: clearInterval],
  clearTimeout: [Function: clearTimeout],
  setInterval: [Function: setInterval],
  setTimeout: [Function: setTimeout] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  queueMicrotask: [Function: queueMicrotask],
  clearImmediate: [Function: clearImmediate],
  setImmediate: [Function: setImmediate] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
```

02. console

- 브라우저에서의 console

```
Object [console] {
  log: [Function: log],
  warn: [Function: warn],
  dir: [Function: dir],
  time: [Function: time],
  timeEnd: [Function: timeEnd],
  timeLog: [Function: timeLog],
  trace: [Function: trace],
  assert: [Function: assert],
  clear: [Function: clear],
  count: [Function: count],
  countReset: [Function: countReset],
  group: [Function: group],
  groupEnd: [Function: groupEnd],
}
```

```
Object [console] {
  table: [Function: table],
  debug: [Function: debug],
  info: [Function: info],
  dirxml: [Function: dirxml],
  error: [Function: error],
  groupCollapsed: [Function: groupCollapsed],
  Console: [Function: Console],
  profile: [Function: profile],
  profileEnd: [Function: profileEnd],
  timeStamp: [Function: timeStamp],
  context: [Function: context]
}
```

03. timer

- setTimeout(callback, 1/1000) : 1/1000 초 이후에 callback 함수 실행
- setInterval(callback, 1/1000) : 1/1000 초 마다 callback 함수 실행
- setImmediate(callback) : callback 즉시 실행
- clearTimeout(id) : setTimeout 취소
- clearInterval(id) : setInterval 취소
- clearImmediate(id) : setImmediate 취소

```
const ftimeout = setTimeout(()=>{
    console.log("1초후 실행"); // 실행 되지 않음 -> 1.5초 이후 취소
}, 2000);

const fInterval = setInterval(()=>{
    console.log("1초 마다 실행"); // 1초 마다 실행 되다가 -> 3.5초 이후 취소
}, 1000);

setTimeout(()=>{
    console.log( " 3.5초 이후 setInterval 정지 " );
    clearInterval(fInterval); //
}, 3500);

setTimeout(()=>{
    console.log("1.5초 이후 setTimeout 정지");
    clearTimeout(ftimeout);
}, 1500);

const fSetImmediate = setImmediate(()=>{
    console.log("즉시 실행 1"); // 즉시 실행
});

const fSetImmediate01 = setImmediate(()=>{
    console.log("즉시 실행 2"); // 바로 실행 취소 됨
});

clearImmediate(fSetImmediate01);
```

4. 내장 객체

04. __filename, __dirname

- 브라우저의 window와 같은 전역 객체로 global을 생략 하고 사용 할 수 있다.

05. process

- 현재 실행 되고 있는 노드 프로세스에 대한 정보
- .exit() : 종료

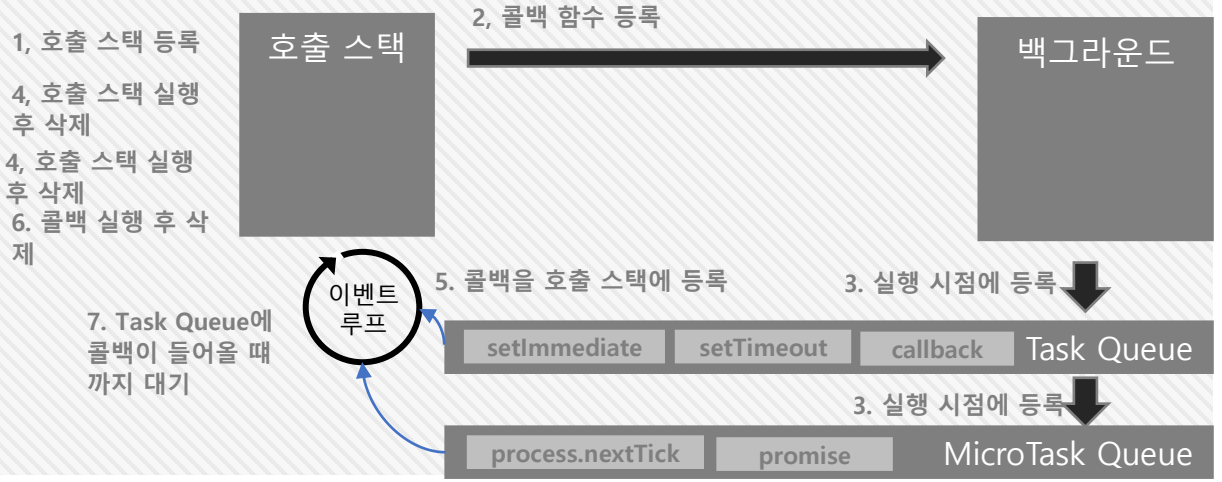
```
console.log('version : ', process.version);  // v14.17.0
console.log('arch : ', process.arch); // x64
console.log('platform : ', process.platform); // win32
console.log('pid : ', process.pid); // 11200
console.log('uptime : ', process.uptime()); // 0.0581393
console.log('execPath : ', process.execPath);
   // C:\Program Files\nodejs\node.exe
console.log('cwd : ', process.cwd());
console.log('cpuUasge : ', process.cpuUsage());
   // { user: 78000, system: 46000 }
console.log('resourceUsage : ', process.resourceUsage());
console.log('memoryUsage : ', process.memoryUsage());


setInterval(()=> {
    console.log(process.cpuUsage());
}, 1000);
```

06. Microtask

- setImmediate, setTimeout 보다 우선 처리 되는 것으로 다른 callback보다 우선 처리 됨
- Resolve된 Promise, process.nextTick
- 주의 사항 : 다른 callback보다 우선 처리 되므로 callback 함수가 실행 되지 않을 수도 있음

```
setImmediate(()=>{
    console.log("Immediate");
});
setTimeout(()=>{
    console.log("Timeout");
}, 0);

process.nextTick(()=>{
    console.log("nextTick");
})

Promise.resolve().then(()=>{
    console.log("Promise");
});
```

Promise
nextTick
Immediate
Timeout



5. 내장 모듈

01. os

- os 모듈 정보를 가지고 온다.

```
import os from 'os';

console.table([{설명: "process.arch", 명령어: "os.arch()", 결과: os.arch()}
               , {설명: "process.platform", 명령어: "os.platform()", 결과: os.platform()}
               , {설명: "운영체제", 명령어: "os.type()", 결과: os.type()}
               , {설명: "운영체제 부팅 이후 시간(초)", 명령어: "os.uptime()", 결과: printTime()}
               , {설명: "컴퓨터의 이름", 명령어: "os.hostname()", 결과: os.hostname()}
               , {설명: "운영체제의 버전", 명령어: "os.release()", 결과: os.release()}
              ]);

console.table([{설명: "홈 디렉토리 경로", 명령어: "os.homedir()", 결과: os.homedir()}
               , {설명: "임시 파일 저장 경로", 명령어: "os.tmpdir()", 결과: os.tmpdir()}
              ]);

console.log(os.cpus());
console.table([{설명: "컴퓨터 core 정보", 명령어: "os.cpus()", 결과: os.cpus().length}
              , {설명: "사용가능한 메모리 정보", 명령어: "os.freemem()", 결과: os.freemem()}
              , {설명: "전체메모리 정보", 명령어: "os.totalmem()", 결과: os.totalmem()}
              ]);

console.log(os.constants);
```

```
function printTime() {
    var ut_sec = os.uptime();
    var ut_min = ut_sec/60;
    var ut_hour = ut_min/60;

    ut_sec = Math.floor(ut_sec);
    ut_min = Math.floor(ut_min);
    ut_hour = Math.floor(ut_hour);

    ut_hour = ut_hour%60;
    ut_min = ut_min%60;
    ut_sec = ut_sec%60;

    return "경과시간: "
           + ut_hour + " 시간 "
           + ut_min + "  분  "
           + ut_sec + "  초";
}
```

02. path

- 폴더와 파일 경로를 쉽게 조작하도록 도와주는 모듈
- sep :윈도우 타입 (C:₩SAMPLE₩)과 POSIX(유닉스 : /home/sample) 타입으로 구분됨
- delimiter : 윈도우 타입 (;), POSIX (:) -> process.env.PATH 구분자를 의미 함

5. 내장 모듈

03. url

- 폴더와 파일 경로를 쉽게 조작하도록 도와주는 모듈
- sep :윈도우 타입 (C:₩SAMPLE₩)과 POSIX(유닉스 : /home/sample) 타입으로 구분됨
- delimiter : 윈도우 타입 (;), POSIX (:) -> process.env.PATH 구분자를 의미 함

```
import url from 'url';

console.log(url);
// WHATWG 방식
Const myNaver = new url.URL(https://post.naver.com/viewer/postView.naver?volumeNo=32179750&memberNo=32165513);
console.log(myNaver);
console.log(url.format(myNaver));
console.log("==================================")

// parse 방식
const myNaverParse = url.parse("https://post.naver.com/viewer/postView.naver?volumeNo=32179750&memberNo=32165513");
console.log(myNaverParse);
console.log(url.format(myNaverParse));
```

04. querystring

- WHATWG 방식의 url 대신 parse

```
{
  Url: [Function: Url],
  parse: [Function: urlParse],
  resolve: [Function: urlResolve],
  resolveObject: [Function: urlResolveObject],
  format: [Function: urlFormat],
  URL: [class URL],
  URLSearchParams: [class URLSearchParams],
  domainToASCII: [Function: domainToASCII],
  domainToUnicode: [Function: domainToUnicode],
  pathToFileURL: [Function: pathToFileURL],
  fileURLToPath: [Function: fileURLToPath]
}
URL {
  href: 'https://post.naver.com/viewer/postView.naver?volumeNo=32179750&memberNo=32165513',
  origin: 'https://post.naver.com',
  protocol: 'https:',
  username: '',
  password: '',
  host: 'post.naver.com',
  hostname: 'post.naver.com',
  port: '',
  pathname: '/viewer/postView.naver',
  search: '?volumeNo=32179750&memberNo=32165513',
  searchParams: URLSearchParams { 'volumeNo' => '32179750', 'memberNo' => '32165513' },
  hash: ''
}
https://post.naver.com/viewer/postView.naver?volumeNo=32179750&memberNo=32165513
==================================
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'post.naver.com',
  port: null,
  hostname: 'post.naver.com',
  hash: null,
  search: '?volumeNo=32179750&memberNo=32165513',
  query: 'volumeNo=32179750&memberNo=32165513',
  pathname: '/viewer/postView.naver',
  path: '/viewer/postView.naver?volumeNo=32179750&memberNo=32165513',
  href: 'https://post.naver.com/viewer/postView.naver?volumeNo=32179750&memberNo=32165513'
}
https://post.naver.com/viewer/postView.naver?volumeNo=32179750&memberNo=32165513
```

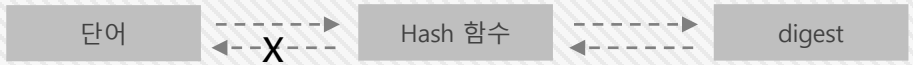
5. 내장 모듈

05. crypto

- 암호화 모듈
- 단방향 암호화 : 복호화 할 수 없는 암호화 방식
- 양방향 암호화 : 키를 이용한 암호화/복호화 방식

#. 단방향 암호화

- Hash 알고리즘 사용 하여 암호화 (md5, sha1, sha256, sha512)
- 인코딩 알고리즘 (base64, hex, latin1 ..)



- PBKDF2 (password-base key derivation function version 2) : 패스워드 기반 키 유로 함수로 Hash 함수, salt, 반복 횟수 등을 지정 하여 다이제스트를 생성


```
import crypto  from "crypto";

const fCrypto = function(str, hash, encode) {
    return crypto.createHash(hash).update(str).digest(encode);
}

console.log("hello :: " , fCrypto("hello", "sha512", "base64"));
```

#. 단방향 암호화

- Key를 이용 해서 대칭 암호화

```
import crypto  from "crypto";

let alogrithm = "aes-256-cbc";
let key = "abcdefghijklmpopqrstuvwxyz123456";
let iv = "1234567890123456";

const fCipher = function( alogrithm, key, iv, str, encoding, outEncoding ) {
    const cry =  crypto.createCipheriv(alogrithm, key, iv);
    let rn = cry.update(str, encoding, outEncoding);
    rn += cry.final(outEncoding);
    return rn;
}

const fDeCipher = function( alogrithm, key, iv, str, encoding, outEncoding ) {
    const dec =  crypto.createDecipheriv(alogrithm, key, iv);
    let rn = dec.update(str,  encoding, outEncoding);
    rn += dec.final(outEncoding);
    return rn;
}

var strClipter = fCipher(alogrithm, key, iv, "안녕하세요", "utf8", "base64");
var strDeCipher = fDeCipher(alogrithm, key, iv, strClipter, "base64", "utf8" );
console.log(strClipter);
console.log(strDeCipher);
```

5. 내장 모듈

06. worker_threads

- Mutti Thread 방식으로 작업 하는 방법
- Web worker : Application의 주 실행 스레드와 별도의 백그라운드 스레드에서 실행 할 수 있는 기술
- Worker는 지정한 javascript file을 다른 전역에서 동작 하는 워커 스레드에서 작동

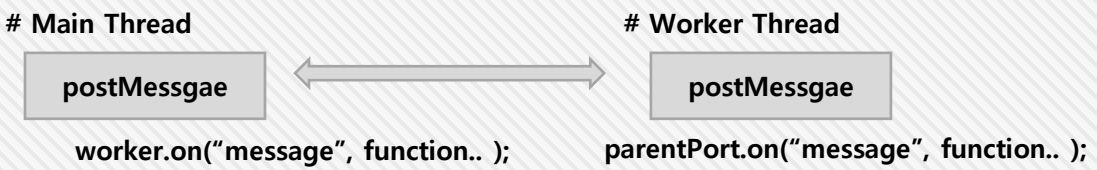
: worker 종류 : DedicatedWorkerGlobalScope : 전용 워커(단일 스크립트에서만 사용 하는 워커) , SharedWorkerGloablScope : 공유 워커 (여러 스크립트에서 공유 하는 워커), 서비스 워커, 오디오 워커

: 워커에서 실행 되지 않는 것 (Dom 조작, window의 일부 메소드 ...)

=> 참고 : https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Functions_and_classes_available_to_workers

 # Web Application

 # Node.js




```
const { Worker, isMainThread, parentPort} = require("worker_threads");

if (isMainThread) {
    const worker = new Worker(__filename);
    worker.on("message", message=>console.log("워커에서 받음 ::", message));
    worker.on("exit", ()=>console.log("워커 종료"));
    worker.postMessage("부모에서 자식에게 보냄");
} else {
    parentPort.on("message", value=> {
        console.log("부모에서 받음 ::", value);
        parentPort.postMessage("자식에서 보냄");
        parentPort.close();
    });
}
결과 ======================================
부모에서 받음 :: 부모에서 게워커 종료
```

5. 내장 모듈

06. worker_threads

- Mutti Thread 방식으로 작업 하는 방법

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


if (isMainThread) {
    const threads = new Set();
    threads.add(new Worker(__filename, {
        workerData: {start: 1}
    }));
    threads.add(new Worker(__filename, {
        workerData: {start: 100}
    }));
    for( let worker of threads) {
        worker.on("message", message=>console.log("워커에서 받음 ::", message));
        worker.on("exit", () => {
            threads.delete(worker);
            if ( threads.size === 0 ) {
                console.log("done....");
            }
        });
    }

} else {
    let data = workerData;
    parentPort.postMessage(data.start + 100);
}

결과 ========================================
워커에서 받음 :: 101
워커에서 받음 :: 200
done....
```

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

function runService(workerData) {
  return new Promise((resolve, reject) => {
    const worker = new Worker('./service.js', { workerData })
    worker.on('message', resolve);
    worker.on('error', reject);
    worker.on('exit', (code) => {
      if (code !== 0) reject(new Error(`Worker stopped with exit code ${code}`));
    });
  });
}

async function run() {
  const result = await runService('world');
  console.log(result);
}

run().catch((err) => console.error(err));

결과 ========================================
{ hello: 'world' }
```

5. 내장 모듈

07. 파일 시스템

- 파일에 대해서 읽기/ 쓰기

```
const fs = require('fs');
fs.readFile('./readme.txt', (err, data)=> {
    if (err) {
        throw err;
    }
    console.log(data); // <Buffer ec 9d b4 ed 8c ……>
    console.log(data.toString()); // 이파일은 예제 파일 입니다
});
```

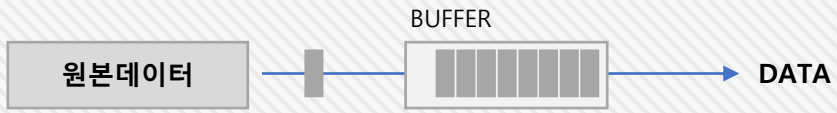
```
const fsPromise = fs.promises;
fsPromise.readFile('./readme.txt')
         .then((data) => {  // 이파일은 예제 파일 입니다.
             console.log("fsPromise.readFile :: ", data.toString());
             fsRead(data.toString());
         })
         .catch((err) => {
            console.log(err);
         });
function fsRead(data) { // 이파일은 예제 파일 입니다.
    console.log("fsRead :: ", data);
    fsWrite(data);
}
function fsWrite(data) {
    fsPromise.writeFile('./readmeOut.txt', data + " :: 파일 쓰기")
             .then(() => {
                return fsPromise.readFile('./readmeOut.txt');
             })
             .then((data)=>{ // 이파일은 예제 파일 입니다. :: 파일 쓰기
             console.log('fsWrite & Read :: ', data.toString());
             })
             .catch((err) => {
             console.log(err);
             });
}
```

```
Async/await ====================================
const fs = require('fs');
let dataReadme01 = fs.readFileSync('./readme.txt');
console.log('sync :: dataReadme01 :: ', dataReadme01.toString());
let dataReadme02 = fs.readFileSync('./readmeOut.txt');
console.log('sync :: dataReadme02 :: ', dataReadme02.toString());

결과
sync :: dataReadme01 ::  이파일은 예제 파일 입니다.
sync :: dataReadme02 ::  이파일은 예제 파일 입니다. :: 파일 쓰기


Promise ========================================
const fsPromise = fs.promises;
fsPromise.readFile('./readme.txt')
    .then((dataReadme01) => {
        console.log('promise :: dataReadme01 :: ', dataReadme01.toString());
        return fsPromise.readFile('./readmeOut.txt');
    })
    .then((dataReadme02) => {
        console.log('promise :: dataReadme02 :: ', dataReadme02.toString());
    })
    .catch((err)=> {
        console.log('err :: ', err);
    });

결과
promise :: dataReadme01 ::  이파일은 예제 파일 입니다.
promise :: dataReadme02 ::  이파일은 예제 파일 입니다. :: 파일 쓰기
```

5. 내장 모듈

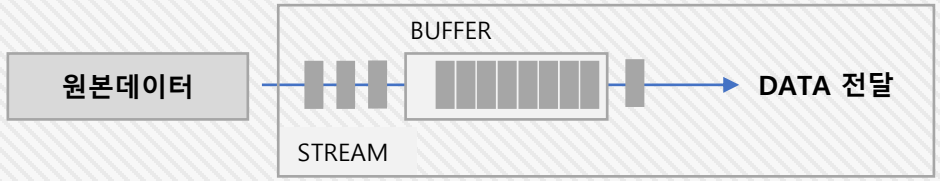
07. buffer

- 파일을 읽어서(네트워크) 메모리에 파일 크기 만큼 공간에 읽은 데이터를 넣어 두는데 이것을 버퍼라 함




```
const buffer = Buffer.from("버퍼에 넣는 정보");
console.log(buffer);
  // <Buffer eb b2 84 ed 8d bc ec 97 90 20 eb 84 a3 eb 8a 94 20 ec a0 95 eb b3 b4>
console.log(buffer.length); // 23
console.log(buffer.toString()); // 버퍼에 넣는 정보
```

 # Stream :: 버퍼의 크기를 작게 만든 후 여러 번으로 나눠 보내는 방식



 # 파이핑

```
const writeStream = fs.createWriteStream('./readmeWrite.txt');
readStream.pipe(writeStream);
writeStream.on('finish', () => {
    console.log('finish :: 종료');
});
```

fs.access
fs.mkdir
fs.open
fs.rename
fs.watch

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


const readStream = fs.createReadStream('./readme.txt', { highWaterMark: 16});
const data = [];

readStream.on('data', (chunk) => {
    data.push(chunk);
    console.log(chunk,  ", length ::" ,  chunk.length);
});

readStream.on('end', () => {
    console.log('end :: ' ,  Buffer.concat(data).toString());
});


readStream.on('error', (err) => {
    console.log('err :: ' ,  err);
});

// 결과
<Buffer ec 9d b4 ed 8c 8c ec 9d bc ec 9d 80 20 ec 98 88> , length :: 16
<Buffer ec a0 9c 20 ed 8c 8c ec 9d bc 20 ec 9e 85 eb 8b> , length :: 16
<Buffer 88 eb 8b a4 2e> , length :: 5
end ::  이파일은 예제 파일 입니다.
```

5. 내장 모듈

08. event (EventEmitter)

- 사용자 이벤트 정의
- addListener, on, emit, once, removeAllListeners, removeListener, off, listenerCount 등의 메서드를 가지고 있음

```
const EventEmitter = require('events');

const userEvent = new EventEmitter();
userEvent.addListener('event', () => {
    console.log("event addListener");
});


userEvent.addListener('event1', (msg) => {
    console.log("event1 addListener :: ", msg);
});

userEvent.emit('event');
userEvent.emit('event1', '안녕');


userEvent.on('eventon', (msg) => {
    console.log("eventon on :: ", msg);
});
userEvent.emit('eventon', '안녕2');
userEvent.emit('eventon', '안녕2');

const listener = (msg, msg1) => {
   console.log("eventon2", msg, msg1 );
};
userEvent.on('listenerEvent', listener);
userEvent.emit('listenerEvent', '안녕2', "우리");
```

```
event addListener
event1 addListener :: 안녕
eventon on :: 안녕2
eventon on :: 안녕2
eventon2 안녕2 우리
```

THANKS

ABACUS

www.iabacus.co.kr

Tel. 82-2-2109-5400

Fax. 82-2-6442-5409