**Stream & Buffer**

We watch YouTube video as stream. Data coming like stream but stream means flow. Flow of data means stream data. Stream is consist of buffer. 01101 1 0011 101011 here all the part is buffer. Chunk of data means buffer. Sequence of stream showed in above.

Stream means flow of data. Flow of data means here buffer flow. Chunk of data make a bundle and create a buffer. And bundle of buffer one by one flow and at last we get data.

Below is a flow diagram:

git remote add origin https://github.com/mkmasudrana806/nodejs-pratice.git

Part3

Part1

Part2

Full Data

Stream

………….. buffer …………………

for example: client want to read a big pdf file. Client make a request to the server wait for while until full data is come. Then client will be read the pdf. If we do not use buffer then full data first load and take many times after that client will be able to see the data.

But if we make a system where we use buffer. When a client will make a request for a big file then we give him data as packet of buffer so that no need to wait for full data. Client instance will be able to read file. No need to wait for the full file. Buffer show the data part by part.

Note: part1, part2 … etc are binary data. We can not see the data directly. Some part make buffer or packet. Then one by one stream. We can not work with these part. but we can work with the buffer. We have access this buffer.

There are two types of stream. Read stream and write stream. Same way will write gradually.

**Read Stream:**

Client

Buffer

dsf

f

git remote add origin https://github.com/mkmasudrana806/nodejs-pratice.git

Data

Chunk of data

After completing buffer, then it come to the client. One by one chunk of data will be fill up to the buffer then this buffer come to the client.

Note: readFileSync, readFile are not streaming way. We need to use createReadStream

createReadStream create stream of buffer. When each buffer is created then we can access this buffer and show to the client. Mainly when one buffer is completed then ourStream will be fired.

const fs = require("fs");

//createReadStream create stream and fire events when one buffer is completed to show the client. so need to work into event to receive the buffer

const ourReadStream = fs.createReadStream(`${\_\_dirname}/big data.txt`, 'utf-8');

ourReadStream.on("data", (chunk) => {

  console.log(chunk);

});

We can pass enconding ‘utf-8’ to the createReadStream function or we can use toString() method inside ourReadStream function.

**Now check wow big data come to the server when a user send a big data to the server.**

const http = require("http");

const server = http.createServer(function (req, res) {

  if (req.url === "/") {

    res.write("<html><head><title>Form</title></head>");

    res.write(

      '<body> <form method="post" action="/process"> <input name="message" /></form> </ body>'

    );

    res.end();

  } else if (req.url === "/process" && req.method === "POST") {

    //in req object we can't get data like req.data. because in streaming way we know that data is come as buffer.after creating one buffer it fire the data event. so we need to use req.on("data", function) like this. because client data is come to the server as streaming way.

    req.on("data", function (chunk) {

      console.log(chunk);

    });

    res.write("Thank you for submitting");

    res.end();

  } else {

    res.write("Not found!");

    res.end();

  }

});

server.listen(5000, () => {

  console.log("Server is listening on port: 5000");

});

We use req.on(“data”, callback) to get data streaming way. But if we want to get full data then we need to use req.on(“end”, callback). Here end means end of the streaming we can get the full data.

Step1: first read one by one buffer and push into body array.

Step2: then goto ‘end’ event. Now convert buffer to string data and do some work.

Here is the code:

const http = require("http");

const server = http.createServer(function (req, res) {

  if (req.url === "/") {

    res.write("<html><head><title>Form</title></head>");

    res.write(

      '<body> <form method="post" action="/process"> <input name="message" /></form> </ body>'

    );

    res.end();

  } else if (req.url === "/process" && req.method === "POST") {

    //in req object we can't get data like req.data. because in streaming way we know that data is come as buffer.after creating one buffer it fire the data event. so we need to use req.on("data", function) like this. because client data is come to the server as streaming way.

    // first we store all the buffered data to the body array

    const body = [];

    req.on("data", function (chunk) {

      body.push(chunk);

    });

    // after end of the streaming the 'end' event will fire

    // now we can work with the data as full data file

    req.on("end", function () {

      console.log("Streaming finished");

      const parsebody = Buffer.concat(body).toString();

      console.log(parsebody);

    });

    res.write("Thank you for submitting");

    res.end();

  } else {

    res.write("Not found!");

    res.end();

  }

});

server.listen(5000, () => {

  console.log("Server is listening on port: 5000");

});

**Note:** NodeJS perform brilliant when big data need to read or write.

Streaming mechanism is the beauty of NodeJS.

**Read and write stream example: req is readable stream and res is writeable stream**

const fs = require("fs");

//createReadStream create stream and fire events when one buffer is completed to show the client. so need to work into event to receive the buffer

// we can pass 'utf-8' encoding to createReadStream

const ourReadStream = fs.createReadStream(`${\_\_dirname}/big data.txt`, "utf-8");

const ourWriteStream = fs.createWriteStream(`${\_\_dirname}/output.txt`, "utf-8");

// or we can use toString() method with chunk

ourReadStream.on("data", (chunk) => {

  // when we get a new buffer of chunk then we write it to our writeStream by 'write' method

  ourWriteStream.write(chunk);

});

**Read and write stream using pipeline:**

const fs = require("fs");

const ourReadStream = fs.createReadStream(`${\_\_dirname}/big data.txt`, "utf-8");

const ourWriteStream = fs.createWriteStream(`${\_\_dirname}/output.txt`, "utf-8");

ourReadStream.pipe(ourWriteStream);

// this line means. pipe er mukhe write stream ase. so pipe die joto buffer asbe. seta ourWriteStream a dhukbe one by one buffer.

// here no need to listen events.