Pontifícia Universidade Católica do Rio Grande do Sul -- Faculdade de Informática   
Redes de Computadores  
  
**Rafael Companhoni** e **Lucka Praisler de Souza**

INTRODUÇÃO

A plataforma Node.js possui um modelo baseado em eventos e callbacks (i.e. métodos vinculados a eventos que são executados quando o evento acontece) . Ela gerencia uma única thread e é sua responsabilidade sinalizar a ocorrência de eventos. Exemplos típicos de eventos ocorrem ao abrir um arquivo, ler seu conteúdo e ao estabelecer uma conexão TCP ou UDP.

Veremos primeiramente como criar um par de programas cliente e servidor que utilizam o protocolo TCP. Neste exemplo vemos que uma conexão é gerada na qual podemos podemos vincular callbacks. Em seguida é demonstrado um par cliente e servidor utilizando o protocolo UDP onde não há uma conexão estabelecida de forma que o servidor apenas 'reage' aos pacotes que são recebidos.

TCP SERVER

Utiliza-se o módulo NPM 'net' para a criação de um servidor através do método createServer(callback). Já na própria criação do objeto é configurada a porta a qual ele receberá dados de seus clientes TCP:

var server = net.createServer(function(conn) {

// ...

}).listen(PORT)

Após a criação do servidor são registrados callbacks para dois eventos que podem ser gerados por este objeto:

* *listening*: Assim que o servidor estiver pronto para receber dados este evento é disparado.
* *error*: Evento gerado em caso de erros. Por exemplo, caso a porta configurada já esteja em uso (erro que é identificado como EADDRINUSE) aguarde-se um intervalo de tempo e tenta-se iniciar o servidor novamente. Caso seja um outro tipo de erro (como tentar acessar a porta 80 que possui privilégios especiais) um mensagem de erro é exibida no console.

Antes de começar a troca de dados, as aplicações cliente e servidor que utilizam o protocolo TCP estabelecem uma conexão através de um procedimento chamado de *handshaking*. A conexão estabelecida é do tipo *full-duplex* já que permite que mensagens sejam enviadas de qualquer uma das partes envolvidas. É também confiável já que o protocolo garante que todos os dados serão enviados e na ordem correta.

O callback utilizado no método createServer possui como um de seus argumentos o objeto *conn* que representa a conexão gerenciada pelo servidor. Este objeto é o que caracteriza um servidor do tipo TCP já que garante que há uma comunicação estabelecida entre o servidor e o cliente. Vincula-se os seguintes eventos a esta conexão:

* *data*: Gerado sempre que o servidor recebe dados de um cliente. Neste caso o arquivo sendo recebido do cliente pode ser enviado em partes que são acumuladas na variável chunks.
* *close*: Gerado quando a conexão com um cliente é encerrada. Neste momento consolida-se um novo objeto do tipo Buffer a partir de chunks. Além disso, utiliza-se um timestamp que será utilizado para gerar o nome do arquivo no servidor.

var net = require('net');

var fs = require('fs');

var moment = require('moment')

const PORT = 8124;

const DIR = './server\_files';

const FILEPATH = DIR + '/server\_';

// cria servidor TCP e vincula eventos a conexão

var server = net.createServer(function(**conn**) {

console.log('conectado!');

// recebe o arquivo do cliente em pedaços

var chunks = [];

conn.on('data', function(data) {

console.log('recebendo dados...');

chunks.push(data);

});

// ao encerrar o envio cria o arquivo no servidor

conn.on('close', function() {

if (!fs.existsSync(DIR)) {

fs.mkdirSync(DIR);

}

const file = Buffer.concat(chunks);

const now = moment().format('DD-MM-YYYY h-mm-ss');

const path = FILEPATH + now + '\_TCP.txt';

fs.writeFile(path, file, function(err) {

if (err)

console.log("erro ao salvar arquivo");

console.log("arquivo salvo com sucesso!");

});

console.log('conexao com o cliente encerrada');

});

}).listen(PORT);

// ao começar a conexão

server.on('listening', function() {

console.log('escutando porta ' + PORT);

});

// tratamento de erros

server.on('error', function(err) {

if (err.code == 'EADDRINUSE') {

console.warn('Endereco em uso, tentando novamente...');

setTimeout(function() {

server.close();

server.listen(PORT);

}, 1000)

}

else {

console.log(err);

}

});

TCP CLIENT

Também é utilizado o módulo NPM 'net' para criar um socket TCP. Este objeto possui os seguintes eventos:

* *connect*: Evento gerado quando o socket conecta-se ao servidor TCP. Neste momento é feita a leitura do arquivo (através do objeto instanciado a partir do módulo 'fs') para um objeto stream -- a este objeto são associados os seguintes eventos/callbacks:
  + Open: quando o stream de leitura é aberto. Neste momento o conteúdo do stream é direcionado para o socket através de um pipe.
  + Error: Apenas exibe o erro no console
* *data*: Quando o cliente recebe dados do servidor
* *close*: Quando a conexão com o servidor é encerrada

var net = require('net');

var fs = require('fs');

const PORT = 8124;

const FILEPATH = './client\_files/client.txt';

var client = new net.Socket();

client.setEncoding('utf8');

// ao conectar com o servidor

client.connect('8124', 'localhost', function(){

console.log('conectado ao servidor');

var fileStream = fs.createReadStream(FILEPATH);

fileStream.on('open',function() {

fileStream.pipe(client);

});

fileStream.on('error', function(err) {

console.log(err);

});

});

// ao receber dados

client.on('data', function(data){

console.log(data);

});

// ao encerrar a conexao

client.on('close', function() {

console.log('conexao encerrada');

});

UDP SERVER

Para a criação de aplicações que envolvem o protocolo UDP utiliza-se o módulo 'dgram'. Assim como nos exemplos anteriores o objeto criado possui o evento *message* que é criado quando um pacote UPD é recebido pelo servidor. No callback utilizado para este evento está a lógica para a criação do arquivo.

Aplicações que se comunicam utilizando o protocolo UDP não estabelecem uma conexão e dessa forma não garantem que os dados serão enviados de forma integral e na ordem correta. Por outro lado, ao dispensar os mecanismos de garantia do protocolo TCP, esta abordagem possibilita o envio de um maior volume de dados e é apropriada para aplicações que toleram uma certa perda dos dados enviados (e.g. telefonia via internet, jogos multiplayer).

var dgram = require('dgram');

var moment = require('moment');

var fs = require('fs');

var server = dgram.createSocket("udp4");

const PORT = 8124;

const DIR = './server\_files';

const FILEPATH = DIR + '/server\_';

// disparado quando um pacote UDP chega neste servidor

server.on("message", function (msg, rinfo) {

if (!fs.existsSync(DIR)) {

fs.mkdirSync(DIR);

}

const now = moment().format('DD-MM-YYYY h-mm-ss');

const path = FILEPATH + now + '\_UDP.txt';

fs.writeFile(path, msg, function (err) {

if (err)

console.log("erro ao salvar arquivo");

console.log("arquivo salvo com sucesso!");

});

});

// quando o servidor for inicializado e estiver pronto para receber pacotes UDP

server.on('listening', function () {

var address = server.address();

console.log('aguardando dados do cliente...');

});

server.bind(PORT);

UDP CLIENT

Lê o arquivo em um objeto *Buffer* e o envia para o servidor através do método send do objeto instanciado a partir do módulo dgram.

var dgram = require('dgram');

var fs = require('fs');

var PORT = 8124;

var HOST = '127.0.0.1';

const FILEPATH = './client\_files/client.txt';

var file = fs.readFileSync(FILEPATH); // buffer

var client = dgram.createSocket('udp4');

client.send(file, 0, file.length, PORT, HOST, function (err, bytes) {

if (err) throw err;

console.log('Arquivo enviado por UDP para ' + HOST + ':' + PORT);

client.close();

});

EXECUTANDO A APLICAÇÃO

Em um computador com a plataforma Node devidamente instalada, acesse o diretório do projeto e instale as dependências com o seguinte comando:

npm install

Abra um terminal e inicialize o servidor TCP:

node tcp\_server.js

Abra um segundo terminal e inicialize o cliente TCP

node tcp\_client.js

Mensagens de status devem ser exibidas no console. O arquivo texto do diretório *client\_files* deve ter sido enviado ao servidor que o copiou para o diretório *server\_files* anexando ao nome o timestamp + ‘TCP’. O mesmo procedimento deve ser realizado para o par cliente/servidor UDP.

REFERÊNCIAS

Powers, Shelley. Learning Node : moving to the server side. Sebastopol, CA: O'Reilly Media, 2016. Print.

Kurose, James F., and Keith W. Ross. *Computer networking : a top-down approach*. Boston: Pearson, 2013. Print.

https://www.hacksparrow.com/tcp-socket-programming-in-node-js.html

http://stackoverflow.com/questions/5970383/difference-between-tcp-and-udp