Table of Contents

1 Envío y recepción de datagramas

El protocolo UDP es un protocolo no orientado a conexión. Cada servidor se deberá “poner a escuchar” en un puerto determinado y los clientes podrán enviar los mensajes a dicho puerto.

Un ejemplo de servidor sería:

// servidor.js

```
'use strict'

const dgram = require('node:dgram');
const server = dgram.createSocket('udp4');

server.on('error', (err) => {
  console.log('Error:\n' + err.stack);
  server.close();
});

server.on('message', (msg, rinfo) => {
  console.log('Mensaje: ' + msg + " recibido de " + rinfo.address + ':' +
rinfo.port);
});

server.on('listening', () => {
  const address = server.address();
  console.log("Escuchando en " + address.address + ":"+ address.port);
});

server.bind(8000);
```

Como se puede ver en el código se usa la biblioteca dgram para crear el socket. Se crea el socket con:

```
const server = dgram.createSocket('udp4');
```

y se pone a escuchar en el puerto 8000:

```
server.bind(8000);
```

Se deben escuchar 3 eventos:

- error: Si se produce un error en la conexión.
- message: Se produce cada vez que llega un mensaje. En la función lambda que se pasa para procesar la respuesta, se tienen como argumentos msg, que es un string, con el mensaje

enviado y rinfo que da información sobre la IP y el puerto desde el que se han enviado el mensaje.
- listening: Se produce cuando el servidor se pone a funcionar.

Un cliente que se conectase y enviase un mensaje al servidor tendría el siguiente aspecto:
// cliente.js

```
'use strict'
const PORT = 8000;
const HOST = '127.0.0.1';

const dgram = require('node:dgram');
const message = new Buffer.from('Hello World');

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

client.send(message, 0, message.length, PORT, HOST, function(err, bytes) {
  if (err) throw err;
  console.log('UDP message sent to ' + HOST +':'+ PORT);
  client.close();
});
```

El cliente sólo puede mandar mensajes de texto al servidor. El mensaje se envía usando un objeto Buffer:

```
const message = new Buffer.from('Hello World');
```

En este caso se va a enviar la cadena de texto “Hello World”.
El método send se usará para enviar un mensaje al servidor:

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

Donde:
- message: es el Buffer con el mensaje a enviar.
- Los dos siguiente argumentos son los las posiciones que se quieren enviar del Buffer (suponga que el Buffer es un array). Lo habitual es enviar todo el Buffer.
- HOST y PORT, dirección y puerto al que se envía el mensaje.
- Por último se debe pasar una función lambda con que se ejecutará cuando se finalice de enviar el mensaje.

Nota: Con el método “close()” se cierra el socket de comunicaciones:

```
client.close();
```

Si se elimina esta línea, el socket permanece abierto.

Curiosidad: Trate de ejecutar dos sesiones del servidor. ¿A qué puede deberse que la segunda sesión genere un error?

1.1 Usando JSON para enviar y recibir los datos

Se pueden usar las funciones “JSON.parse(msg)” y “JSON.stringify(a)” para interpretar texto en JSON o transformar variables en JSON.

Por ejemplo:

// cliente2.js

```
'use strict'
const PORT = 8000;
const HOST = '127.0.0.1';
```

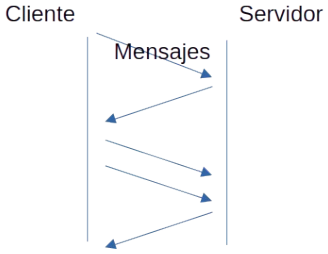
```
const dgram = require('node:dgram');

const a = {'nombre': 'pepe', 'telefono': 555}

const message = new Buffer.from(JSON.stringify(a));

const client = dgram.createSocket('udp4');
client.send(message, 0, message.length, PORT, HOST, function(err, bytes) {
  if (err) throw err;
  console.log('UDP message sent to ' + HOST +':'+ PORT);
  client.close();
});
```

// servidor2.js

```
'use strict'

const dgram = require('node:dgram');
const server = dgram.createSocket('udp4');

server.on('error', (err) => {
  console.log('Error:\n' + err.stack);
  server.close();
});

server.on('message', (msg, rinfo) => {
  console.log('Mensaje: ' + msg + " recibido de " + rinfo.address + ':' +
rinfo.port);
  const a = JSON.parse(msg);
  console.log(a.nombre);
  console.log(a.telefono);
});

server.on('listening', () => {
  const address = server.address();
  console.log("Escuchando en " + address.address + ":"+ address.port);
});

server.bind(8000);
```

1.2 Comunicaciones cliente-servidor

En los ejemplos anteriores el servidor no responde al cliente y el cliente no espera una respuesta del servidor. Pero se puede tener una comunicación en la que cliente y servidor intercambian mensajes:



El cliente también puede escuchar el evento “message” y el servidor puede contestar al cliente usando al método “send”. Por ejemplo, el siguiente servidor responde al cliente con el mismo mensaje que le envía (eco):

// servidor3.js

```
'use strict'

const dgram = require('node:dgram');
const server = dgram.createSocket('udp4');

server.on('error', (err) => {
  console.log('Error:\n' + err.stack);
  server.close();
});

server.on('message', (msg, rinfo) => {
  console.log('Mensaje: ' + msg + " recibido de " + rinfo.address + ':' +
rinfo.port);
  const message = new Buffer.from(msg);
  server.send(message, 0, message.length, rinfo.port, rinfo.address,
function(err, bytes) {
    if (err) throw err;
    console.log('UDP message sent to ' + rinfo.address + ':' + rinfo.port);
  });
});

server.on('listening', () => {
  const address = server.address();
  console.log("Escuchando en " + address.address + ":"+ address.port);
});

server.bind(8000);
```

El cliente de ejemplo, escucha el evento “message” para recibir el mensaje del servidor y responder al mismo:

// cliente3.js

```
'use strict'
const PORT = 8000;
const HOST = '127.0.0.1';

const dgram = require('node:dgram');
const message = new Buffer.from('Hello World');

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

let contador = 0;

client.on('message', (msg, rinfo) => {
  console.log('Mensaje: ' + msg + " recibido de " + rinfo.address + ':' +
rinfo.port);
  const message = new Buffer.from(msg + contador);
  client.send(message, 0, message.length, rinfo.port, rinfo.address,
function(err, bytes) {
    if (err) throw err;
    console.log('UDP message sent to ' + rinfo.address + ':' + rinfo.port);
    contador++;
    if(contador > 5) client.close();
  });
});

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

```
  if (err) throw err;
  console.log('UDP message sent to ' + HOST +':'+ PORT);
});
```

Nota: En todo momento se puede cerrar el socket de comunicaciones usando el método “close”:

```
if(contador > 5) client.close();
```

2 Conexiones TCP

Este caso es muy similar al de los datagramas.

Para crear el servidor se usará:

// servidor-tcp1.js

```
'use strict'
const net = require("net");

const port = 5000;

const server = net.createServer((socket) => {
    console.log("Client connected");

    socket.on("data", (data) => {
        const strData = data.toString();
        console.log('Recibido: ' + strData);
        socket.write(strData);
    });

    socket.on("end", () => {
        console.log("Client disconnected");
    });

    socket.on("error", (error) => {
        console.log('Socket Error: ' + error.message);
    });
});

server.on("error", (error) => {
  console.log('Socket Error: ' + error.message);
});

server.listen(port, () => {
    console.log("TCP socket server is running on port: " + port);
});
```

Ahora los eventos a escuchar son:

- data: Se produce cada vez que se recibe un conjunto de datos del cliente.
- end: Se produce cuando se cierra la conexión con el cliente.
- error: Se produce cuando hay errores en la comunicación.

Para mandar datos al cliente se usará el método “write()” del objeto socket.

Con el método “listen” del objeto server, se indica el puerto en el que el cliente debe escuchar al servidor.

// cliente-tcp1.js

```
'use strict'
const net = require("net");

const host = "127.0.0.1";
const port = 5000;

const client = net.createConnection(port, host, () => {
  console.log("Connected");
  client.write('Hello World');
});

let contador = 0;

client.on("data", (data) => {
  console.log("Received: " + data);
  if(contador > 5) client.end();
  else client.write("" + contador++);
});

client.on("error", (error) => {
  console.log("Error: " + error.message);
});

client.on("close", () => {
  console.log("Connection closed");
});
```

Los eventos del cliente son muy similares a los del servidor. Sólo destacar que las conexiones pueden ser cerradas con el método “end”.

2.1 Usando JSON para el envío y recepción de información

Al igual que en el caso de los datagramas, el formato JSON se puede usar para el envío y recepción de información. Por ejemplo, el cliente se puede modificar de la siguiente forma:

// cliente-tcp2.js

```
'use strict'
const net = require("net");

const host = "127.0.0.1";
const port = 5000;

const client = net.createConnection(port, host, () => {
  console.log("Connected");
  const a = {'usuario': 'Pepe', 'contador' : 0};
  client.write(JSON.stringify(a));
});

client.on("data", (data) => {
  console.log("Received: " + data);
  const a = JSON.parse(data);
  a.contador += 1;
```

```
  if(a.contador > 5) client.end();
  else client.write(JSON.stringify(a));
});

client.on("error", (error) => {
  console.log("Error: " + error.message);
});

client.on("close", () => {
  console.log("Connection closed");
});
```