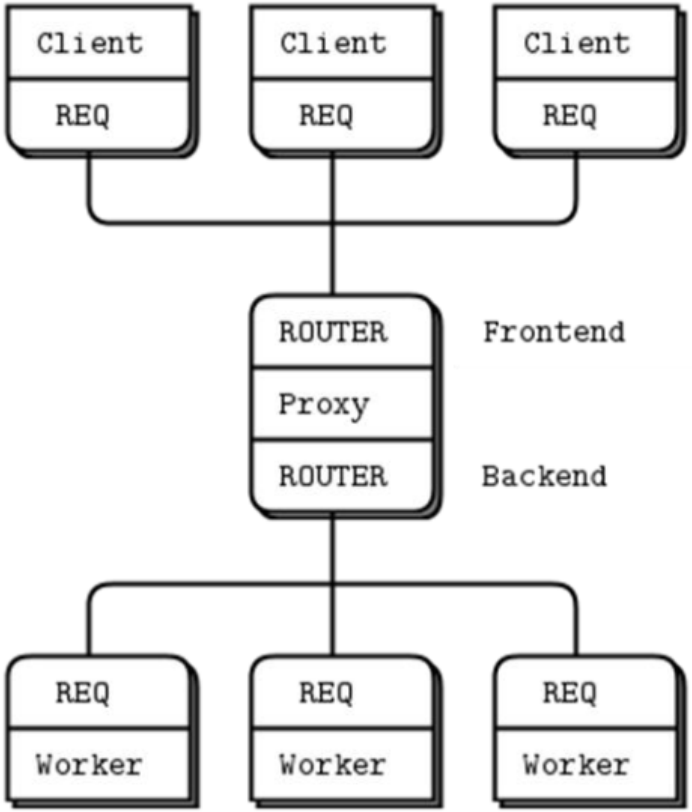
# Prácticas TSR
# Laboratorio 2:sesiones 2,3

Juan Sánchez (jsanchez@dsic.upv.es)

Despacho 2D08

# 2.2 Broker entre clientes y servidores

- Pruebas básicas: un cliente, un trabajador y un broker



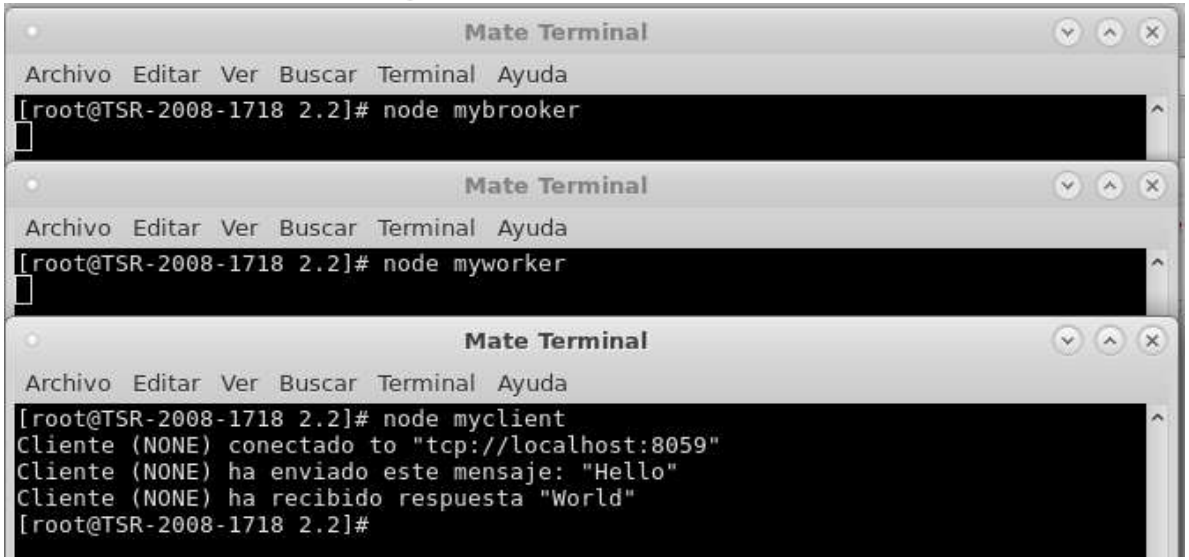

http://zguide.zeromq.org/page:all#A-Load-Balancing-Message-Broker

# 2.2 Broker entre clientes y servidores

- Copie el código que se encuentra al final del documento, el listado de la página 45 contiene un error en la línea 59, quite el "." que se encuentra después de "="

```
59:         var m = . [workerID,'',nextClient.id,''].concat(nextClient.msg);
```

- La ejecución sobre la misma máquina sin argumentos de línea de comandos genera eco únicamente en el cliente



# 2.4.1 Modo verbose

- Añadimos el código de la página 24 al trabajador y al broker para poder activar el modo verbose.
- En la primera prueba no utilizaremos argumentos en la línea de comandos, para ello utilice en el broker.

```
var verbose = args[2]|| true
```

- Mientras en el cliente y el worker hacemos algo parecido, no olvide añadir respectivamente en el cliente y en el worker.

```
requester.identity= 'Cliente 1';

responder.identity= 'Worker 1';
```

- En la siguiente sección lo pasaremos en la línea de comandos

# 2.4.1 Modo verbose

- El código resultante en el broker debe ser algo parecido a:

```
var workers = [];
// Array of pending clients.
var clients = [];

frontend.bindSync('tcp://*:'+fePortNbr);
backend.bindSync('tcp://*:'+bePortNbr);
// Nueva funcion de eco en consola
function sendToWorker(msg) {
if (verbose) {
     console.log('Sending client (%s) request to worker (%s) through backend.',msg[2], msg[0]);
     aux.showMessage(msg);
     }
backend.send(msg);
}
```

- Sustituya en el código original backend.send() por sendToWorker()

# 2.4.1 Modo verbose

- Arrancamos en este orden: broker, worker y client

```
[root@TSR-2008-1718 2.2]# node mybroker
Sending client (Cliente 1) request to worker (Worker 1) through backend.
     Segment 0: Worker 1
     Segment 1:
     Segment 2: Cliente 1
     Segment 3:
     Segment 4: Hello
```

```
[root@TSR-2008-1718 2.2]# node myworker
Worker (NONE) connected to tcp://localhost:8060
Worker (NONE) has sent its first connection message: "id"
Worker (NONE) has received request Hello from client (Cliente 1)
Worker (NONE) has sent its reply (World)
```

```
[root@TSR-2008-1718 2.2]# node myclient
Cliente (NONE) conectado to "tcp://localhost:8059"
Cliente (NONE) ha enviado este mensaje: "Hello"
Cliente (NONE) ha recibido respuesta "World"
[root@TSR-2008-1718 2.2]#
```

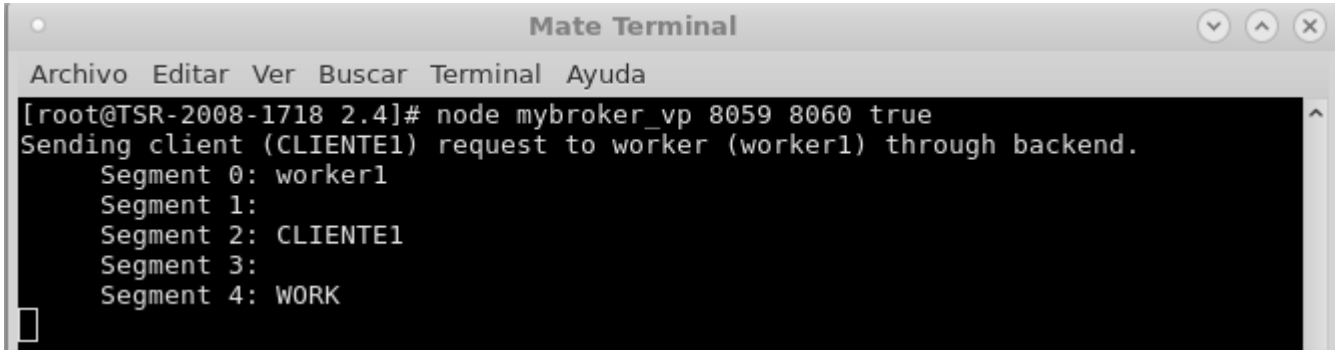
# 2.4.2 Parametrizando el código

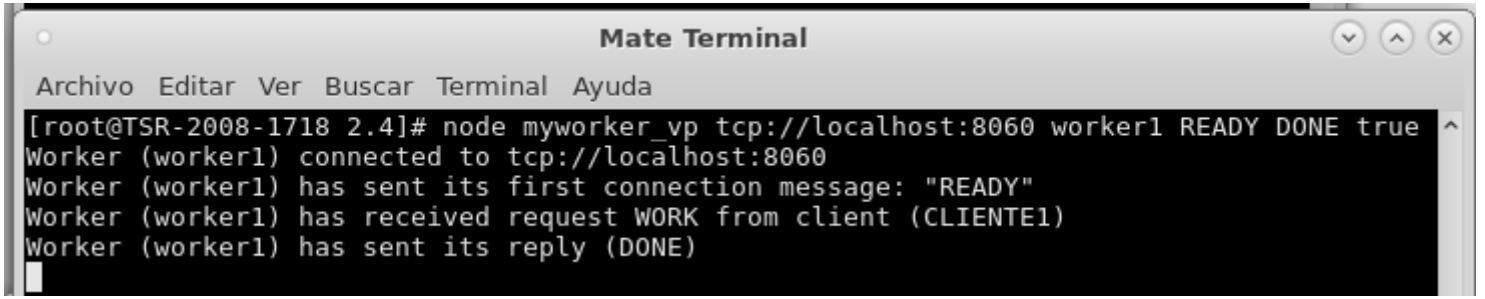
- Añadimos parámetros al client y al worker

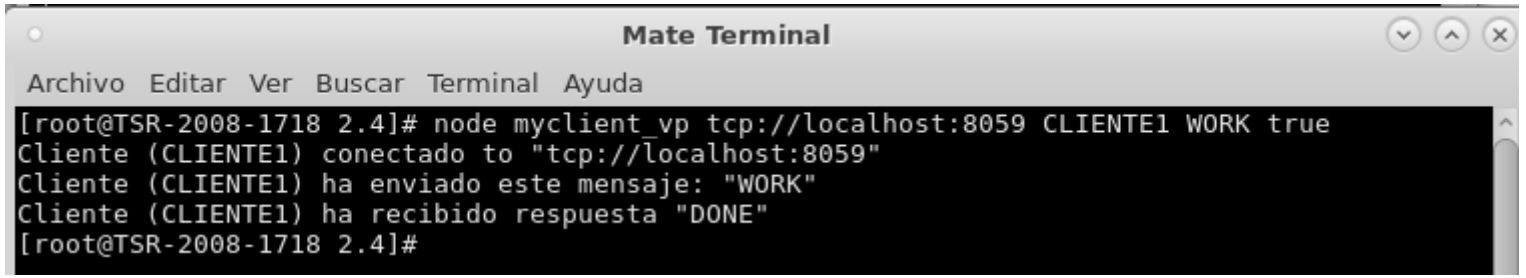

- Adicionalmente añadimos el parámetro verbose

# 2.4.2 Parametrizando el código

- Arrancando los programas



# Pruebas

- Un cliente y dos servidores

```
[root@TSR-2008-1718 2.4]# node mybroker_vp 8059 8060 true
Sending client (CLIENTE1) request to worker (worker1) through backend.
     Segment 0: worker1
     Segment 1:
     Segment 2: CLIENTE1
     Segment 3:
     Segment 4: WORK
```

```
[root@TSR-2008-1718 2.4]# node myworker_vp tcp://localhost:8060 worker1 READY DONE true
Worker (worker1) connected to tcp://localhost:8060
Worker (worker1) has sent its first connection message: "READY"
Worker (worker1) has received request WORK from client (CLIENTE1)
Worker (worker1) has sent its reply (DONE)
```

```
[root@TSR-2008-1718 2.4]# node myworker_vp tcp://localhost:8060 worker2 READY DONE true
Worker (worker2) connected to tcp://localhost:8060
Worker (worker2) has sent its first connection message: "READY"
```

```
[root@TSR-2008-1718 2.4]# node myclient_vp tcp://localhost:8059 CLIENTE1 WORK true
Cliente (CLIENTE1) conectado to "tcp://localhost:8059"
Cliente (CLIENTE1) ha enviado este mensaje: "WORK"
Cliente (CLIENTE1) ha recibido respuesta "DONE"
[root@TSR-2008-1718 2.4]#
```

# Pruebas

- Dos clientes y un servidor

```
[root@TSR-2008-1718 2.4]# node mybroker_vp 8059 8060 true
Sending client (CLIENTE1) request to worker (worker1) through backend.
     Segment 0: worker1
     Segment 1:
     Segment 2: CLIENTE1
     Segment 3:
     Segment 4: WORK1
Sending client (CLIENTE2) request to worker (worker1) through backend.
     Segment 0: worker1
     Segment 1:
     Segment 2: CLIENTE2
     Segment 3:
     Segment 4: WORK2
```

```
[root@TSR-2008-1718 2.4]# node myworker_vp tcp://localhost:8060 worker1 READY DONE true
Worker (worker1) connected to tcp://localhost:8060
Worker (worker1) has sent its first connection message: "READY"
Worker (worker1) has received request WORK1 from client (CLIENTE1)
Worker (worker1) has sent its reply (DONE)
Worker (worker1) has received request WORK2 from client (CLIENTE2)
Worker (worker1) has sent its reply (DONE)
```

```
[root@TSR-2008-1718 2.4]# node myclient_vp tcp://localhost:8059 CLIENTE1 WORK1 true
Cliente (CLIENTE1) conectado to "tcp://localhost:8059"
Cliente (CLIENTE1) ha enviado este mensaje: "WORK1"
Cliente (CLIENTE1) ha recibido respuesta "DONE"
[root@TSR-2008-1718 2.4]#
```

```
[root@TSR-2008-1718 2.4]# node myclient_vp tcp://localhost:8059 CLIENTE2 WORK2 true
Cliente (CLIENTE2) conectado to "tcp://localhost:8059"
Cliente (CLIENTE2) ha enviado este mensaje: "WORK2"
Cliente (CLIENTE2) ha recibido respuesta "DONE"
[root@TSR-2008-1718 2.4]#
```

# Pruebas

- Dos clientes y tres servidores, esta prueba la puede obviar

# 2.5/2.6 Más implementación y pruebas completas

- Modificar el broker para que muestre al terminar las peticiones atendidas por cada trabajador.
- La condición de terminación es que no queden más clientes por atender o bien al pulsar control-c
- Listado en el broker de trabajadores utilizados

```
// detectado control-c
process.on('SIGINT', function() {
    console.log('Detectado control-c, terminación del broker');
    for (var trabajador in workersUsados)
          console.log("El trabajador: " + trabajador + " se ha utilizado " + workersUsados[trabajador]);
    process.exit();
    });
```

- workerUsados es un map, vea siguiente transparencia

# 2.5/2.6 Más implementación y pruebas completas

- En el broker añadimos

```
// Array of available workers.
var workers = [];
// Array of pending clients.
var clients = [];

var workersUsados = {};
function storeWorker(nombre, veces)
{
    workersUsados[nombre] = veces;
}
```

- En la primera conexión de un trabajador

```
backend.on('message', function() {
    var args = Array.apply(null, arguments);
    // If this is an initial ID message from a new worker,
    // save its identity in the "workers" array...
    if (args.length == 3) {
        // It may happen that some clients are already waiting
        // for available workers. If so, serve one of them!
        storeWorker(args[0],0);
        if (!processPendingClient(args[0]))
        // Otherwise, save this worker as an availabe one.
        workers.push(args[0]);
    // ...otherwise, return the message to the appropriate
    // client.
    } else {
        // Save the worker identity.
```

# 2.5/2.6 Más implementación y pruebas completas

- Y cuando termina el trabajo de un worker (esto es continuación del código anterior)

```
} else {
    // Save the worker identity.
    var workerID = args[0];
    // Remove the first two slots (id+delimiter)
    // from the array.
    args = args.slice(2);
    // Send the resulting message to the appropriate
    // client. Its first slot holds the client connection
    // ID.
    frontend.send(args);
    // Incrementar el número de veces que se ha usado el worker
    if (verbose) console.log('Devolviendo respuesta ' + workerID +' a ' + args);

    workersUsados[workerID]++;
```

- Para que los trabajadores terminen después de un intervalo de tiempo añadimos en el worker.

```
responder.send(connText);
if (verbose)
    console.log('Worker (%s) has sent its first connection message: "%s"', myID, connText);

setTimeout(function() {
    console.log('Terminando trabajador.' + myID);
    process.exit(0);
}, 100000);
```

# 2.5 Más implementación y pruebas completas

- Utilizaremos sendos scripts para crear 100 clientes y 10 trabajadores.
- Para los clientes: `bash crearclientes.sh`

crearclientes.sh × | creartrabajadores.sh × | mybroker_vp.js × | myclient_vp.js × | my

```
1  # For loop 100 veces
2  # crear clientes
3  for i in {1..100}
4  do
5      node myclient_vp tcp://localhost:8059 "cliente $i" work true &
6  done
7  wait;
```

- De igual forma para se hace para crear los trabajadores
- Desde la línea de comandos arrancamos el broker, los trabajadores y los clientes

```
[root@TSR-2008-1718 2.5]# node mybroker_vp 8059 8060 true
```

```
[root@TSR-2008-1718 2.5]# bash creartrabajadores.sh
```

```
[root@TSR-2008-1718 2.5]# bash crearclientes.sh
```

# 2.5 Más implementación y pruebas completas

- La salida de ejecución muestra para el broker, después de terminar todo el procesamiento

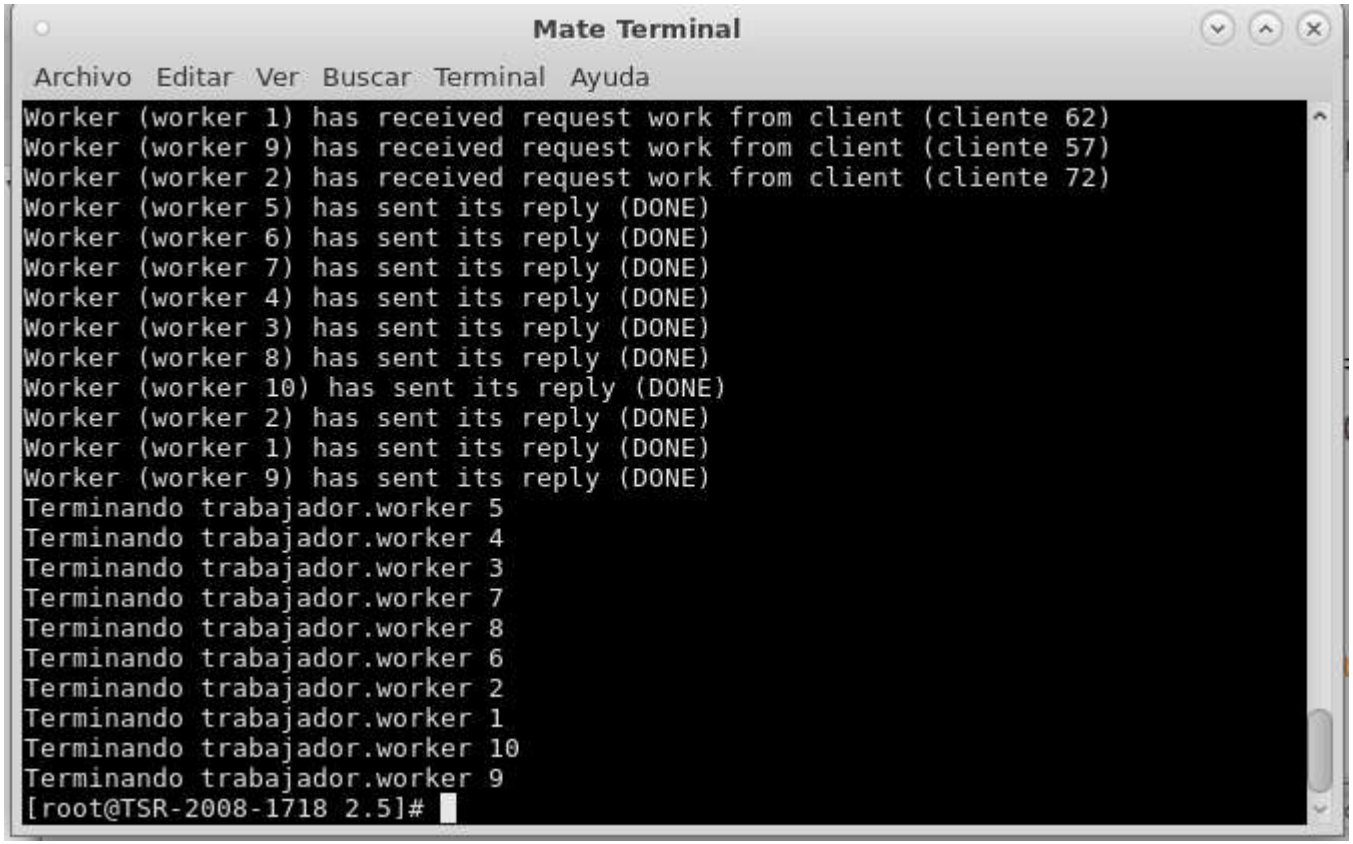
Mate Terminal

Archivo Editar Ver Buscar Terminal Ayuda

```
Sending client (cliente 83) request to worker (worker 10) through backend
     Segment 0: worker 10
     Segment 1:
     Segment 2: cliente 83
     Segment 3:
     Segment 4: work
Devolviendo respuesta worker 9 a cliente 97,,DONE
Sending client (cliente 57) request to worker (worker 9) through backend.
     Segment 0: worker 9
     Segment 1:
     Segment 2: cliente 57
     Segment 3:
     Segment 4: work
Devolviendo respuesta worker 5 a cliente 78,,DONE
Devolviendo respuesta worker 6 a cliente 66,,DONE
Devolviendo respuesta worker 7 a cliente 88,,DONE
Devolviendo respuesta worker 4 a cliente 67,,DONE
Devolviendo respuesta worker 3 a cliente 84,,DONE
Devolviendo respuesta worker 8 a cliente 18,,DONE
Devolviendo respuesta worker 10 a cliente 83,,DONE
Devolviendo respuesta worker 2 a cliente 72,,DONE
Devolviendo respuesta worker 1 a cliente 62,,DONE
Devolviendo respuesta worker 9 a cliente 57,,DONE
```

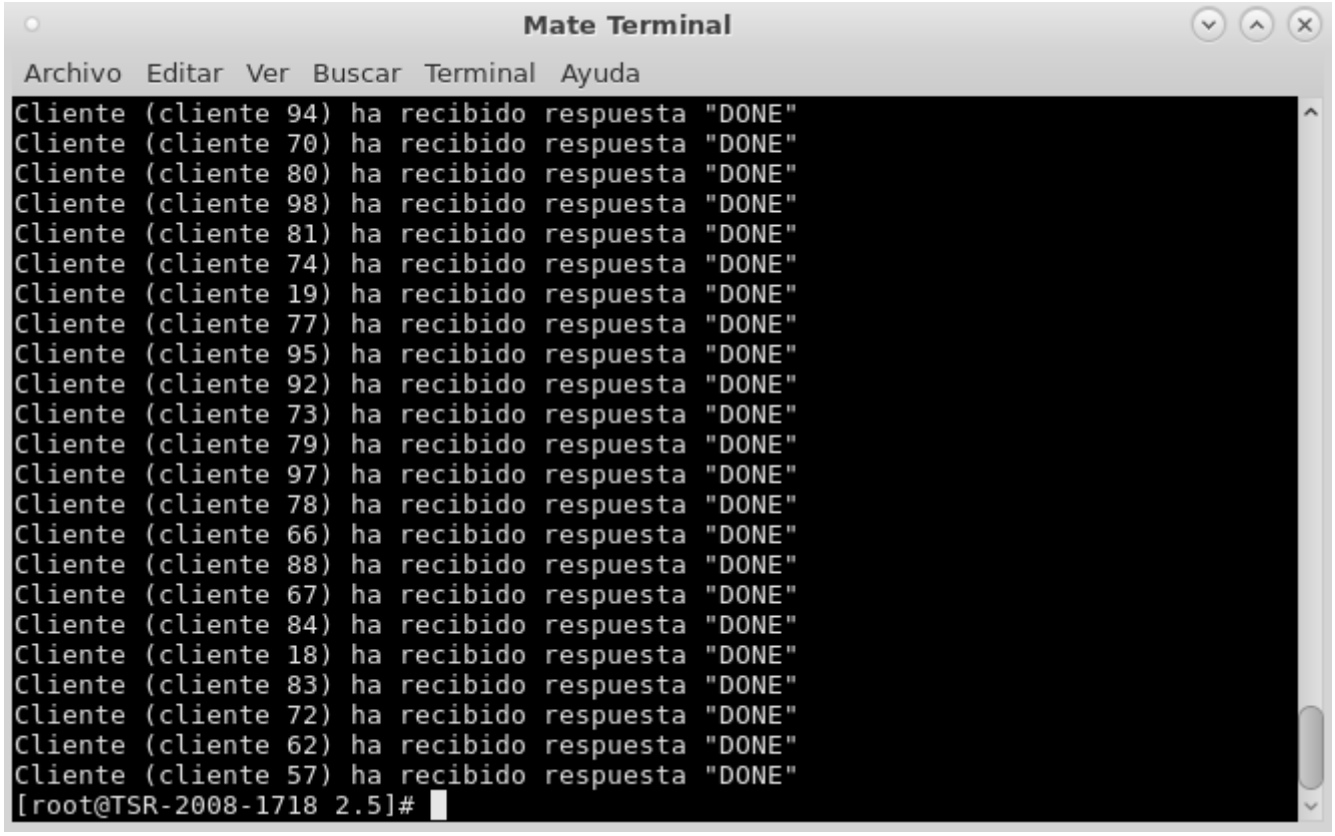
# 2.5 Más implementación y pruebas completas

- Para los trabajadores



# 2.5 Más implementación y pruebas completas

- Para los clientes



# 2.5/2.6 Más implementación y pruebas completas

- Por último pulsamos control-c sobre el broker

```
Devolviendo respuesta worker 1 a cliente 62,,DONE
Devolviendo respuesta worker 9 a cliente 57,,DONE
^CDetectado control-c, terminación del broker
El trabajador: worker 6 se ha utilizado 10
El trabajador: worker 5 se ha utilizado 10
El trabajador: worker 4 se ha utilizado 10
El trabajador: worker 3 se ha utilizado 10
El trabajador: worker 7 se ha utilizado 10
El trabajador: worker 8 se ha utilizado 10
El trabajador: worker 2 se ha utilizado 10
El trabajador: worker 1 se ha utilizado 10
El trabajador: worker 10 se ha utilizado 10
El trabajador: worker 9 se ha utilizado 10
[root@TSR-2008-1718 2.5]#
```

# 3. Esquema Router-Router

- 3.1 Tipos de trabajo: los clientes solicitan un tipo de trabajo y los trabajadores indican el tipo de trabajo que pueden resolver.
- El código de la página 31 puede ser ejecutado tal como está, puede omitir esta parte y pasar a la siguiente sección

# 3.1 Esquema ROUTER-ROUTER

- 3.1 Tipos de trabajo. Los tipos de trabajo se conocen con anterioridad, al invocar al broker
- `node broker_vp 8059 8060 true A B C D E F`
- Reutilice el código de mybrooker_vp (añadiendo ahora el manejo de los tipos de trabajadores y de tipos de trabajo para clientes), myclient_vp (añadiendo el tipo de trabajo en el que está interesado), myworker_vp (añadiendo el tipo de trabajo que puede realizar).

# 3.1 Esquema ROUTER-ROUTER

```
node myclient_vp tcp://localhost:8059 "cliente 1" work A true &
```

- Hay que modificar las peticiones en el cliente

```
// myclient_vp in Node.js
// sección 3.1 boletín Práctica 2
// cliente que pide trabajos de un tipo determinado

var zmq = require('zmq'),
    requester = zmq.socket('req');

// Command-line arguments.
var args = process.argv.slice(2);
// Get those arguments.
var brokerURL = args[0] || 'tcp://localhost:8059'; //end point
var myID = args[1] || 'NONE';  // identidad del cliente
var myMsg = args[2] || 'Hello'; // texto petición servicio
var classID = args[3] || 'B';   // tipo de servicio o de trabajador
var verbose = args[4] || true;  // modo verbose

if (myID!= 'NONE') requester.identity = myID;


requester.connect(brokerURL);
if (verbose)
console.log('Cliente (%s) conectado to "%s"', myID, brokerURL);

requester.on("message", function(msg) {
  if (verbose)
  console.log('Cliente (%s) ha recibido respuesta "%s"', myID, msg.toString());
  process.exit(0);
});

requester.send([myMsg, classID]);
if (verbose)
console.log('Cliente (%s) ha enviado este mensaje: %s con clase %s', myID, myMsg,classID);
```

# 3.1 Esquema ROUTER-ROUTER

- Cambios en el código del broker_vp (pag 31 y 32)
  - Estructura para almacenar clientes y trabajadores

```
// Array of available workers.
var workers = [];
// Array of pending clients.
var clients = [];

// toma los tipos de trabajos desde la línea de comandos.
const classIDs = process.argv.slice(5);
    for (var i in classIDs){
        workers[classIDs[i]]=[];
        clients[classIDs[i]]=[];
    }
```

  - Las peticiones de clientes llevan ahora como último argumento el tipo de trabajo solicitado (A,B..)

```
frontend.on('message', function() {
// Note that separate message parts come as function arguments.
    var args = Array.apply(null, arguments);
    var classID = args.pop(); // extrae el tipo de trabajo que viene como último argumento
    // Check whether there is any available worker.
    if (workers[classID].length > 0) {
        var myWorker = workers[classID].shift();
        var m = [myWorker,''].concat(args);
    sendToWorker(m);
    } else
        clients[classID].push( {id: args[0],msg: args.slice(2)});
    });
```

# 3.1 Esquema ROUTER-ROUTER

- La función de procesamiento pendiente debe recibir el tipo de trabajo

```
function processPendingClient(workerID, classID) {
    if (clients[classID].length>0) {
    var nextClient = clients[classID].shift();
    var m = [workerID,'',nextClient.id,''].concat(nextClient.msg);
    sendToWorker(m);
    return true;
    } else
      return false;
}
```

- El código que maneja los mensajes del backend debe también modificarse

```
backend.on('message', function() {
    var args = Array.apply(null, arguments);
    console.log('Devolviendo respuesta desde el backend');
    var classID = args.pop(); // tipo de trabajo debe venir como último argumento.
    if (args.length == 3) {
        storeWorker(args[0],0);
        if (!processPendingClient(args[0], classID))
        workers[classID].push(args[0]);
    } else {
            var workerID = args[0];
            args = args.slice(2);
            frontend.send(args);
            if (verbose) console.log('Devolviendo respuesta ' + workerID +' a ' + args);
            workersUsados[workerID]++;
            if (!processPendingClient(workerID, classID))
                workers[classID].push(workerID);
            }
});
```

# 3.1 Esquema ROUTER-ROUTER

- Para probar la solución utilizaremos dos ficheros de scripts: crearclientes.sh, creartrabajadores.sh

```
# crear clientes

node myclient_vp tcp://localhost:8059 "cliente 1" work A true &
node myclient_vp tcp://localhost:8059 "cliente 2" work A true &
node myclient_vp tcp://localhost:8059 "cliente 3" work B true &
node myclient_vp tcp://localhost:8059 "cliente 4" work C true &
node myclient_vp tcp://localhost:8059 "cliente 5" work D true &
wait;
```

```
# crear trabajadores

node myworker_vp tcp://localhost:8060 "worker 1" READY DONE  A true &
node myworker_vp tcp://localhost:8060 "worker 2" READY DONE  B true &
node myworker_vp tcp://localhost:8060 "worker 3" READY DONE  C true &
node myworker_vp tcp://localhost:8060 "worker 4" READY DONE  D true &
node myworker_vp tcp://localhost:8060 "worker 5" READY DONE  A true &
wait;
```

# 3.1 Esquema ROUTER-ROUTER

- Arrancando en este orden, mybroker_vp, clientes y trabajadores

```
[root@TSR-2008-1718 3.1]# node mybroker_vp 8059 8060 true A B C D E F
```

```
[root@TSR-2008-1718 3.1]# bash crearclientes.sh
Cliente (cliente 4) conectado to "tcp://localhost:8059"
Cliente (cliente 3) conectado to "tcp://localhost:8059"
Cliente (cliente 5) conectado to "tcp://localhost:8059"
Cliente (cliente 1) conectado to "tcp://localhost:8059"
Cliente (cliente 2) conectado to "tcp://localhost:8059"
Cliente (cliente 2) ha enviado este mensaje: work con clase A
Cliente (cliente 4) ha enviado este mensaje: work con clase C
Cliente (cliente 3) ha enviado este mensaje: work con clase B
Cliente (cliente 5) ha enviado este mensaje: work con clase D
Cliente (cliente 1) ha enviado este mensaje: work con clase A
Cliente (cliente 1) ha recibido respuesta "DONE"
Cliente (cliente 2) ha recibido respuesta "DONE"
Cliente (cliente 3) ha recibido respuesta "DONE"
Cliente (cliente 4) ha recibido respuesta "DONE"
Cliente (cliente 5) ha recibido respuesta "DONE"
```

# 3.1 Esquema ROUTER-ROUTER

- Para los trabajadores

```
^C[root@TSR-2008-1718 3.1]# bash creartrabajadores.sh
Worker (worker 1) con tipo de trabajo A connected to tcp://localhost:8060
Worker (worker 3) con tipo de trabajo C connected to tcp://localhost:8060
Worker (worker 5) con tipo de trabajo A connected to tcp://localhost:8060
Worker (worker 4) con tipo de trabajo D connected to tcp://localhost:8060
Worker (worker 2) con tipo de trabajo B connected to tcp://localhost:8060
Worker (worker 5)  del tipo A has sent its first connection message: "READY"
Worker (worker 1)  del tipo A has sent its first connection message: "READY"
Worker (worker 2)  del tipo B has sent its first connection message: "READY"
Worker (worker 3)  del tipo C has sent its first connection message: "READY"
Worker (worker 4)  del tipo D has sent its first connection message: "READY"
Worker (worker 5) has received request work from client (cliente 1)
Worker (worker 1) has received request work from client (cliente 2)
Worker (worker 3) has received request work from client (cliente 4)
Worker (worker 2) has received request work from client (cliente 3)
Worker (worker 4) has received request work from client (cliente 5)
Worker (worker 5) has sent its reply (DONE)
Worker (worker 1) has sent its reply (DONE)
Worker (worker 2) has sent its reply (DONE)
Worker (worker 3) has sent its reply (DONE)
Worker (worker 4) has sent its reply (DONE)
Terminando trabajador.worker 5
Terminando trabajador.worker 1
Terminando trabajador.worker 2
Terminando trabajador.worker 3
Terminando trabajador.worker 4
[root@TSR-2008-1718 3.1]#
```

# 3.1 Esquema ROUTER-ROUTER

- Finalmente para el broker

```
Devolviendo respuesta worker 3 a cliente 4,,DONE,
Devolviendo respuesta desde el backend
     Segment 0: worker 4
     Segment 1:
     Segment 2: cliente 5
     Segment 3:
     Segment 4: DONE
     Segment 5:
     Segment 6: D
Devolviendo respuesta worker 4 a cliente 5,,DONE,
^CDetectado control-c, terminación del broker
El trabajador: worker 5 se ha utilizado 1
El trabajador: worker 1 se ha utilizado 1
El trabajador: worker 2 se ha utilizado 1
El trabajador: worker 3 se ha utilizado 1
El trabajador: worker 4 se ha utilizado 1
[root@TSR-2008-1718 3.1]#
```

# 3.2 Latido

- Detección de caídas de los trabajadores: utilizaremos el código mybroker2.js de las páginas 34 y 35 del boletín
- Ahora no empleamos tipos de trabajadores ni tipos de peticiones de clientes.
- Los trabajadores myworker_vp y myclient o myclient_vp son los utilizados en la sección 2.5/2.6 de esta presentación

# 3.2 Latido

- Las pruebas están cambiadas con respecto al orden del boletín, primero probaremos el broker con latido y posteriormente el broker sin latido

# 3.2 Latido

- myworker_vp

```
// myworker server in NodeJS
// 3.2 Latido, igual que en sección 2.5/2.6
var zmq = require('zmq') , responder = zmq.socket('req');

// Get the command-line arguments, if any.
var args = process.argv.slice(2);
var backendURL = args[0] || 'tcp://localhost:8060'; // endpoint
var myID = args[1] || 'NONE';  //identidad del trabajador
var connText = args[2] || 'READY'; // disponibilidad del trabajador
var replyText = args[3] || 'DONE'; // mensaje atención servicio
var verbose = args[4] || false;

if (myID != 'NONE') responder.identity = myID;
// Connect to the broker backend socket.
responder.connect(backendURL);
if (verbose)
    console.log('Worker (%s) connected to %s', myID, backendURL);
// Process each incoming request.
responder.on('message', function(client, delimiter, msg) {
    if (verbose) {
        console.log('Worker (%s) has received request %s from client (%s)',
            myID, msg.toString(), client);
    }
    setTimeout(function() {
        responder.send([client,'',replyText]);
        if (verbose)
            console.log('Worker (%s) has sent its reply (%s)',
                myID, replyText);
    }, 1000);
});
```

# 3.2 Latido

- Continuación myworker_vp

```
responder.send(connText);
if (verbose)
    console.log('Worker (%s) has sent its first
connection message: "%s"', myID, connText);

setTimeout(function() {
    console.log('Terminando trabajador.' + myID);
    process.exit(0);
}, 100000);
```

# 3.2 Latido

- Con respecto al cliente: myclient o myclient_vp

```
// myclient_vp in Node.js
// sección 2.4 boletín Práctica 2

var zmq = require('zmq'),
    requester = zmq.socket('req');

// Command-line arguments.
var args = process.argv.slice(2);
// Get those arguments.
var brokerURL = args[0] || 'tcp://localhost:8059'; //end point
var myID = args[1] || 'NONE';  // identidad del cliente
var myMsg = args[2] || 'Hello'; // texto petición servicio
var verbose = args[3] || true;  // modo verbose

if (myID!= 'NONE') requester.identity = myID;
requester.connect(brokerURL);
if (verbose)
    console.log('Cliente (%s) conectado to "%s"', myID, brokerURL);

requester.on("message", function(msg) {
    if (verbose)
        console.log('Cliente (%s) ha recibido respuesta "%s"', myID, msg.toString());
    process.exit(0);
});

requester.send(myMsg);
if (verbose)
    console.log('Cliente (%s) ha enviado este mensaje: "%s"', myID, myMsg);
```

# 3.2 Latido: pruebas

- Probaremos la combinación resultante, antes añadiremos un pequeño fragmento de código que nos indica el PID de cada proceso.
- En mybroker2.js

```
frontend.bindSync('tcp://*:'+fePortNbr);
backend.bindSync('tcp://*:'+bePortNbr);
var process = require('process');
if (process.pid) {
  console.log('Identificador del proceso (pid) del broker ' + process.pid);
}
```

- De igual forma hacemos en myworker

# 3.2 Latido: pruebas

- Arrancaremos el broker2 y 3 trabajadores con el siguiente script (esto difiere del contenido del boletín)

```
## prueba latencia, script de creación
node mybroker2 8059 8060 true &
node myworker_vp 'tcp://localhost:8060' Worker1 READY DONE true &
node myworker_vp 'tcp://localhost:8060' Worker2 READY DONE true &
node myworker_vp 'tcp://localhost:8060' Worker3 READY DONE true &
wait
```

- Desde la línea de comandos: bash pruebalatencia.sh

```
[root@TSR-2008-1718 3.2]# bash pruebalatencia.sh
Identificador del proceso (pid) trabajador Worker1 12090
Identificador del proceso (pid) trabajador Worker3 12092
Identificador del proceso (pid) trabajador Worker2 12091
Worker (Worker2) connected to tcp://localhost:8060
Identificador del proceso (pid) del broker 12089
Worker (Worker1) connected to tcp://localhost:8060
Worker (Worker1) has sent its first connection message: "READY"
Worker (Worker3) connected to tcp://localhost:8060
Worker (Worker3) has sent its first connection message: "READY"
Worker (Worker2) has sent its first connection message: "READY"
```

- Ya tenemos los pid de cada proceso, matamos el proceso 12090

# 3.2 Latido: pruebas

- Ahora creamos 5 clientes con un script, igual que antes.

```
# crear clientes

node myclient_vp tcp://localhost:8059 "cliente 1" work true &
node myclient_vp tcp://localhost:8059 "cliente 2" work true &
node myclient_vp tcp://localhost:8059 "cliente 3" work true &
node myclient_vp tcp://localhost:8059 "cliente 4" work true &
node myclient_vp tcp://localhost:8059 "cliente 5" work true &
wait;
```

- El intercambio de mensajes es ahora

```
[root@TSR-2008-1718 3.2]# bash pruebalatencia.sh
Identificador del proceso (pid) trabajador Worker1 12090
Identificador del proceso (pid) trabajador Worker3 12092
Identificador del proceso (pid) trabajador Worker2 12091
Worker (Worker2) connected to tcp://localhost:8060
Identificador del proceso (pid) del broker 12089
Worker (Worker1) connected to tcp://localhost:8060
Worker (Worker1) has sent its first connection message: "READY"
Worker (Worker3) connected to tcp://localhost:8060
Worker (Worker3) has sent its first connection message: "READY"
Worker (Worker2) has sent its first connection message: "READY"
Worker (Worker2) has received request work from client (cliente 5)
Worker (Worker3) has received request work from client (cliente 3)
Worker (Worker2) has sent its reply (DONE)
Worker (Worker2) has received request work from client (cliente 2)
Worker (Worker3) has sent its reply (DONE)
Worker (Worker3) has received request work from client (cliente 1)
Worker (Worker2) has sent its reply (DONE)
Worker (Worker2) has received request work from client (cliente 4)
Worker (Worker3) has sent its reply (DONE)
Worker (Worker2) has sent its reply (DONE)
```

# 3.2 Latido: más pruebas

- Ahora lanzamos el broker_vp utilizando un script, este broker no contempla el latido y lo compararemos con el caso anterior (está cambiado el orden de pruebas con respecto al boletín)
- Utilizamos el script

```
## prueba latencia, script de creación
## broker original del apartado 2.5 no contempla la latencia.
node mybroker_vp 8059 8060 true &
node myworker_vp 'tcp://localhost:8060' Worker1 READY DONE true &
node myworker_vp 'tcp://localhost:8060' Worker2 READY DONE true &
node myworker_vp 'tcp://localhost:8060' Worker3 READY DONE true &
wait
```

# 3.2 Latido: más pruebas

- Matamos el proceso del primer trabajador
- Arrancamos los 5 clientes y observamos el comportamiento, un cliente se queda sin respuesta

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
[root@TSR-2008-1718 3.2]# kill 13100
[root@TSR-2008-1718 3.2]# bash crearclientes.sh
Cliente (cliente 1) conectado to "tcp://localhost:8059"
Cliente (cliente 4) conectado to "tcp://localhost:8059"
Cliente (cliente 3) conectado to "tcp://localhost:8059"
Cliente (cliente 3) ha enviado este mensaje: "work"
Cliente (cliente 5) conectado to "tcp://localhost:8059"
Cliente (cliente 2) conectado to "tcp://localhost:8059"
Cliente (cliente 1) ha enviado este mensaje: "work"
Cliente (cliente 4) ha enviado este mensaje: "work"
Cliente (cliente 5) ha enviado este mensaje: "work"
Cliente (cliente 2) ha enviado este mensaje: "work"
Cliente (cliente 2) ha recibido respuesta "DONE"
Cliente (cliente 1) ha recibido respuesta "DONE"
Cliente (cliente 4) ha recibido respuesta "DONE"
Cliente (cliente 5) ha recibido respuesta "DONE"
```

```
Sending client (cliente 5) request to worker (Worker2) through backend.
     Segment 0: Worker2
     Segment 1:
     Segment 2: cliente 5
     Segment 3:
     Segment 4: work
Worker (Worker2) has received request work from client (cliente 5)
Worker (Worker3) has sent its reply (DONE)
Devolviendo respuesta Worker3 a cliente 4,,DONE
Worker (Worker2) has sent its reply (DONE)
Devolviendo respuesta Worker2 a cliente 5,,DONE
Terminando trabajador.Worker3
Terminando trabajador.Worker2
```

# 3.3 Equilibrado de carga

- Lo veremos en la sesión 4.