TSR: Ejercicio 3

Estos programas son un cliente y un servidor. El servidor recibe un argumento desde la línea de órdenes y retorna una respuesta que incluye dos segmentos. El primero mantiene el contenido de la solicitud y el segundo proporciona la respuesta. Esa respuesta es el valor de la solicitud multiplicado por el valor recibido como argumento desde la línea de órdenes. Una vez el cliente recibe cada respuesta, muestra el contenido de ese mensaje.

Act3-client.js

```
var zmq = require('zmq');
var req = zmq.socket('req');
var counter = 1;
const interval = 1000;

req.bindSync('tcp://*:8000');

function sendMsg() {
  req.send(counter++);
}

function handler(segm1,segm2) {
  console.log('Request: %d, reply: %s',
              segm1,segm2);
  if (segm1>99) process.exit(0);
}

req.on('message', handler);

setInterval(sendMsg, interval);
```

Act3-server.js

```
var zmq = require('zmq');
var rep = zmq.socket('rep');
var args = process.argv.slice(2);
if (args.length < 1) {
  console.error('Please provide an argument!');
  process.exit(1);
}
rep.connect('tcp://127.0.0.1:8000');

rep.on('message', function(msg) {
  rep.send([msg,msg*args[0]]);
});
```

Responda justificadamente a las siguientes preguntas en una hoja aparte:

1. Justifique si estos programas desarrollan correctamente lo que se ha descrito en el enunciado. En caso de que no sea así, explique qué sentencias son incorrectas y proporcione una versión correcta de ellas. (1 punto)
2. Explique cuántos mensajes son enviados por el cliente y con qué ratio de envío (esto es, cuántos mensajes en cada unidad de tiempo). (1 punto)
3. Describa si existe algún límite para el número de clientes que podrán ser iniciados. (1 punto)
4. Describa si existe algún límite para el número de servidores que podrán ser iniciados. (1 punto)
5. Justifique si, con el código mostrado arriba, clientes y servidores pueden ser ubicados en ordenadores diferentes. (1 punto)
6. Describa qué será mostrado por el cliente durante sus primeros 10 segundos de ejecución si esta secuencia de acciones ocurre en un ordenador determinado: (1) Un cliente es iniciado en el instante 0 seg.; (2) en el instante 2.9 seg., un servidor es iniciado utilizando "**node Act3-server 2**"; (3) en el instante 5.2 seg., un segundo servidor es iniciado utilizando "**node Act3-server 5**". (3 puntos)
7. Explique cómo se tendrán que iniciar múltiples servidores para generar la siguiente salida en el cliente (2 puntos):

```
Request: 1, reply: 3
Request: 2, reply: 6
Request: 3, reply: 9
Request: 4, reply: 8
Request: 5, reply: 25
Request: 6, reply: 18
Request: 7, reply: 14
Request: 8, reply: 40
Request: 9, reply: 27
Request: 10, reply: 20
...
```

Solución comentada

Soluciones

1. Ambos programas desarrollan correctamente lo que se ha descrito en el enunciado. Act3-server.js deja los argumentos recibidos en la variable **args**. En el bloque entre las líneas 4 y 7 comprueba si se ha recibido al menos un argumento y muestra un mensaje de error en caso de que no sea así. En ese mismo bloque, mediante **process.exit(1)**, se cierra el proceso. La respuesta construida por el servidor se genera en la última sentencia del programa. Allí se pasa como argumento de la llamada al método **send()** un vector con dos componentes. Cada componente será un segmento del mensaje de respuesta. El primer segmento es el valor recibido (obsérvese que es el argumento del *listener* para el evento **'message'** del socket utilizado por el servidor) y el segundo se obtiene multiplicando el primer segmento por el valor recibido como argumento desde la línea de órdenes. El comportamiento del cliente se describe con detenimiento en la respuesta de la siguiente pregunta. También corresponde a lo descrito en el enunciado.
2. Con la última sentencia del programa cliente se establece que la función **sendMsg** será invocada una vez por segundo. Dentro de esa función **sendMsg** se observa que en cada invocación se incrementa el valor de un contador. Ese mismo valor, antes del incremento, es enviado como contenido de un mensaje al servidor. Posteriormente, en la función **handler**, observamos que se muestra un mensaje por pantalla, reportando cuál ha sido el valor enviado y cuál el valor obtenido como respuesta. Si el valor del primer segmento supera 99 unidades, el cliente termina. Al estar utilizando un patrón REQ-REP esto implica que el cliente generará 100 mensajes y antes de emitir el centésimo primero (pues la respuesta al centésimo llega antes), finalizará.
3. El cliente utiliza **bindSync** empleando un número de puerto fijo (el 8000). Como no pueden realizarse dos llamadas **bind** con éxito sobre un mismo puerto local, solo se podrá generar un solo cliente.
4. El servidor se conecta al puerto utilizado por el cliente en la máquina local. Nada impide que varios procesos se conecten a un mismo URL. Por tanto, no existe ningún límite sobre el número de servidores a iniciar.
5. No pueden ubicarse en ordenadores diferentes pues el programa servidor solo puede conectarse a un URL local. Cliente y servidor deben estar en un mismo ordenador para poder interactuar.
6. El cliente genera un mensaje por segundo. Eso implica que en el instante 1seg se enviará la primera solicitud al servidor o servidores. En ese momento todavía no hay ningún servidor conectado, por lo que ese mensaje de petición se queda en el buffer de salida. Ocurre lo mismo en el instante 2 seg. En el instante 2.9 seg. se conecta el primer servidor. Con ello, el primer mensaje llega al servidor y es respondido. Como ambos procesos están en la misma máquina, costará unos pocos milisegundos (aunque dependerá de la carga que haya en el equipo y la estrategia de planificación utilizada por el sistema operativo) entregar esa respuesta. En ese momento se mostrará "**Request: 1, reply: 2**" en pantalla. Al haberse entregado la primera respuesta, el socket REQ del cliente será capaz de transmitir el segundo mensaje que se dejó en su cola de salida. A los pocos milisegundos se mostrará "**Request: 2, reply: 4**" en pantalla. En los instantes 3 seg, 4 seg y 5 seg se mostrarán respectivamente los mensajes "**Request: 3, reply: 6**", "**Request: 4, reply: 8**" y "**Request: 5, reply: 10**". A partir del segundo 6 tenemos dos servidores conectados al socket REQ del cliente. Cuando eso sucede, el socket REQ distribuye los mensajes enviados circularmente entre los procesos conectados a él. Por ello, se observará lo siguiente en los cinco mensajes que faltaba distribuir en los segundos 6, 7, 8, 9 y 10: "**Request: 6, reply: 12**" (multiplicado por 2), "**Request: 7, reply: 35**" (multiplicado por 5), "**Request: 8, reply: 16**" (multiplicado por 2), "**Request: 9, reply: 45**" (multiplicado por 5) y "**Request: 10, reply: 20**" (multiplicado por 2).

Solución comentada

7. Tendremos que empezar observando qué factor se ha utilizado para realizar cada multiplicación. Así, vemos que los mensajes 1, 2, 3, 6 y 9 se han multiplicado por 3, mientras que los mensajes 4, 7 y 10 se han multiplicado por 2 y los mensajes 5 y 8 se han multiplicado por 5. Por tanto, al final había tres servidores, pero los tres primeros mensajes fueron atendidos por un mismo servidor (el que multiplicaba por 3).

   Hay varias formas de generar ese resultado. Comentaremos algunas de las posibles.

   Un posible ejemplo sería:

   - Entre los instantes 0 seg y 3.5 segundos se ha tenido que lanzar con "**node Act3-server 3**" un primer servidor que multiplicaba las solicitudes por 3. Si ese servidor se lanzó a partir de los tres segundos, ha debido tener tiempo para ser el único servidor existente capaz de atender los tres primeros mensajes mantenidos en la cola de salida del cliente.
   - Antes del instante 4" pero después del instante 2.1", respetando las restricciones comentadas en el punto anterior, se ha generado otro servidor con la orden "**node Act3-server 2**". Con ello, este segundo servidor ha sido capaz de recibir el cuarto mensaje emitido por el cliente.
   - Antes del instante 5" y después de haber lanzado ese segundo servidor, se ha llegado a lanzar un tercer servidor con la orden "**node Act3-server 5**". Por ejemplo, tanto el segundo como el tercer servidor pudieron ser lanzados de manera consecutiva poco después de los 2" de haber iniciado el cliente, si asumimos que el primer servidor fue lanzado antes de los 2".