Servidores Escalables

TSR 2021. Grupo B, Juansa Sendra

Servidores Escalables

- Para implantar servicios distribuidos utilizamos el modelo cliente/servidor
  - La parte servidor
    - Recibe peticiones, las procesa, devuelve respuestas
    - Puede solicitar servicios a otros servidores
    - Es escalable si puede aceptar otras peticiones antes de completar la petición en curso
- Para implementar un servidor escalable son posibles dos paradigmas:
  - Servidor concurrente
  - Servidor asincrónico

Servidor concurrente

- Múltiples hilos que se ejecutan de forma solapada en el tiempo
  - Cada petición se atendida por un hilo diferente
- Estado compartido. Todos los hilos comparten un estado global $\rightarrow$ requiere mecanismos de control de concurrencia para garantizar atomicidad
- Es el modelo habitual (Java, .NET) $\rightarrow$ estudiado en CSD

Servidor asincrónico

- Dirigido por eventos. (nodeJS, Async .NET)
- Programa = conjunto de acciones preparadas para responder a diferentes tipos de eventos (guarda $\rightarrow$ acción, ..., guarda $\rightarrow$ acción)
- Si la llegada de un evento hace cierta una guarda, la acción asociada se activa
  - Si no estamos ejecutando otra cosa, ejecutamos la acción
  - Si estamos ejecutando otra cosa, guardamos la acción en una cola
    - Cola de eventos (guarda las acciones activadas pero pendientes de ejecución)

Servidor asincrónico.- ejemplo (broker asincrónico)

```
const zmq = require('zeromq') // utilizara el middleware 0MQ bajo el nombre zmq
let sc = zmq.socket('router') // socket para conexión con clientes (frontend)
let sw = zmq.socket('dealer') // socket para conexión con workers  (backend)
sc.bind('tcp://*:9998') // punto de conexión de clientes
sw.bind('tcp://*:9999') // punto de conexión de workers
sc.on('**message', // guarda (si llega solicitud de un cliente)
        (...m) => {sw.send(m)}) // acción (la reenvia a un worker)
sw.on('message', // guarda (si llega respuesta de un worker)
        (...m) => {sc.send(m)}) // acción (la reenvia al cliente)
```

Comparamos ventajas/inconvenientes de los dos modelos

- Servidor concurrente
  - ↑ Cada hilo se puede suspender por separado
  - ↓ Gestionar el estado compartido requiere control concurrencia → suspensión
  - ↓ Complejidad: difícil implantar sin errores, difícil razonar/justificar corrección
- Servidor asincrónico
  - ↑ Evita la complejidad de gestionar estado compartido → escala mejor
  - ↑ Modelo más próximo a la forma real de trabajo → dirigido por eventos
    - facilita el razonamiento sobre la corrección del código
  - ↓ Tiene que considerarse el orden de activación (orden en la cola de eventos)
    - Necesita una gestión adecuada del estado al implantar las acciones
  - ↓ Todo el entorno tiene que ser asincrónico (ej. también los servicios del SONIDO)

Servidores escalables.- Cuestiones

Plantéate las siguientes cuestiones

- El ejemplo de servidor asincrónico (broker asincrónico) no parece tener ningún bucle. Cuando acaba el programa?
- El servidor asincrónico dispone solo de un hilo. ¿Qué pasa si mientras ejecuta código llega un evento?