Programación Concurrente

Práctica 7: Mensajes

Ejercicio 1. Se desea definir un servidor que recibe dos tipos de mensajes de un único cliente: "sigue" y "cuenta".

a) Cada vez que el servidor recibe un mensaje "cuenta" debe mostrar por pantalla la cantidad de mensajes "sigue" que recibió desde la última vez que recibió "cuenta".

b) Modificar la solución para hacer que el servidor en lugar de mostrar por pantalla el número correspondiente, lo envíe al cliente y sea este último el que lo muestre por pantalla. Dar también el código para el cliente.

Ejercicio 2. Se desea implementar un servidor que recibe mensajes de (exactamente) dos clientes distintos y muestra por pantalla la concatenación de un par de mensajes (uno de cada cliente). Cada cliente continuamente envía un mensaje y espera una cantidad aleatoria de tiempo. El servidor recibe los mensajes de los clientes y por cada par de mensajes recibidos muestra por pantalla su concatenación.

Ejercicio 3. Se desea definir un servidor que funcione como una variable compartida a la que los clientes pueden acceder mediante oportunos mensajes. Proponer una solución.

Ejercicio 4. Considerar un servidor de trimming que recibe un String y responde con el resultado de aplicar la función trim al valor recibido.

a) Dar una solución para el caso en que el servidor resuelve solicitudes de un único cliente a la vez.

b) Extender la solución para soportar un número no acotado de clientes (i.e., con esquema de conexión).

Ejercicio 5. Se desea definir un servidor que provea el siguiente comportamiento: por cada cliente que se conecta debe generar un número pseudoaleatorio entero en el rango [0,10]. Los clientes deben intentar adivinar el número generado por el servidor. Para ello envían sucesivamente mensajes conteniendo un número. Cada mensaje es contestado por el servidor indicando si el cliente acertó o no. Notar que el cliente no puede enviar un nuevo mensaje hasta tanto el servidor no le conteste el anterior.

Ejercicio 6. Se desea definir un servicio de codificación y transmisión de información llamado T. El servicio T recibe la información que un cliente C quiere enviar a un servicio remoto R. El servicio T es el encargado de codificar cada mensaje recibido desde C y de reenviarlo a R. Puede asumir que cuenta con la función `codificar` que se encarga de codificar un mensaje.

```
String codificar(String mensaje)
```

Se solicita:

a) Dar el código para el servicio T.

b) Suponer ahora que el servicio T debe permitir a un cliente reenviar la información a distintos servicios remotos. Para ello el cliente debe indicar al servicio T quién será el destinatario de los mensajes que se enviarán a continuación (los destinatarios estarán identificados por un canal). Por ejemplo, el cliente puede comenzar indicando al servicio T que los mensajes que se enviarán a continuación deberán ser reenviados al servicio remoto 1, luego todos los mensajes que envíe a continuación el cliente, y hasta tanto este no notifique a T que desea cambiar el destinatario, deberán ser codificados y reenviados al servicio remoto 1.

Asumir que el cliente siempre comenzará indicando al servicio T cual es el primer servicio remoto al que se deberán reenviar los mensajes. Sin embargo, no puede realizar ninguna suposición sobre la cantidad de mensajes que se enviarán a cada servidor remoto.

Dar una solución para el servicio T que exhiba el comportamiento descripto.

c) Suponer ahora que para codificar cada mensaje el servicio T necesita contar con una clave nueva que es generada por otro servicio de codificación K. Esta clave será recibida por el servicio T a través de un canal dedicado. Luego, el servicio T deberá codificar cada mensaje del cliente utilizando una clave nueva que recibirá desde el servicio K. Para realizar la codificación con una clave puede utilizar la siguiente función:

```
String codificar(String mensaje, String clave)
```

Dar una implementación para el servicio T descripto anteriormente.

Ejercicio 7. Se cuenta con un servicio S que permite hacer cálculos numéricos computacionalmente costosos. S recibe cada pedido de cálculo a realizar en un mensaje sobre el canal 1. El servicio toma el requerimiento, lo procesa y envía la respuesta a través del canal 2. Es decir, un cliente C envía un mensaje sobre el canal 1 y espera la respuesta en el canal 2. Como consecuencia de un mal diseño el cliente puede enviar varios pedidos de cálculo sin esperar a que los anteriores hayan sido procesados, y por otro lado S no garantiza el funcionamiento correcto cuando recibe un nuevo mensaje sobre el canal 1 y aún está procesando un mensaje anterior. Por este motivo, se desea implementar otro servicio proxy P que intermedie la interacción entre C y S. La intención es que el cliente C no interactuará directamente con S (es decir, no usará los canales 1 y 2), en su lugar interactuará sólo con P sobre los canales 3 y 4. La idea es que P reenvíe a S los mensajes que recibe de C y viceversa pero asegurando que S nunca recibirá un nuevo mensaje sobre el canal 1 hasta que no haya concluido de procesar el pedido anterior.

a) Dar una implementación para P. La solución propuesta debe funcionar asumiendo que S no cambia y en C sólo se renombran los canales 1 y 2 por 3 y 4.

b) Suponer ahora que S puede manejar a lo sumo N mensajes concurrentes de manera correcta. Extender la solución propuesta anteriormente para P de manera tal de maximizar la concurrencia. Asumir que S produce los resultados respetando el orden en que llegan los pedidos siempre y cuando haya a lo sumo N pedidos concurrentes. ¿Puede asumir que su solución también está devolviendo los resultados en orden?

c) Extender la implementación de P para aprovechar el hecho de que existen dos copias de S (una que usa los canales 1 y 2 y la otra que usa los canales 11 y 22). P debe aprovechar las dos copias de S a fin de maximizar la concurrencia. (Variante difícil: considerar que al cliente se le deben retornar los resultados en el orden en que hizo los requerimientos.)

Ejercicio 8. Se desea implementar usando mensajes una simulación llamada el Juego de la Vida. El Juego de la Vida procede en un universo en forma de grilla bidimensional en donde cada celda puede estar viva o muerta. La simulación progresa generación tras generación siguiendo los cambios en las celdas de la grilla a partir de una configuración inicial. Cada celda interactúa con sus 8 celdas vecinas siguiendo las siguientes reglas:

- Una celda viva con estrictamente menos de dos vecinos vivos muere de soledad.
- Una celda viva con dos o tres vecinos vivos, continua viviendo.
- Una celda viva con estrictamente más de tres vecinos vivos, muere por sobrepoblación.
- Una celda muerta con exactamente tres vecinos vivos se convierte en una celda viva por efecto de reproducción.

Implemente:

a) Un proceso `Timer` que toma por parámetro: (i) un canal para la recepción de pedidos de registro; (ii) un entero que indica la frecuencia (en mili-segundos) con la que se generarán "ticks"; y (iii) un entero que indica la cantidad mínima de clientes registrados necesaria para empezar a generar "ticks". Una vez alcanzada la cantidad mínima de clientes registrados, el `Timer` debe por siempre generar mensajes "tick" con una cadencia (aproximada) dada por parámetro, enviado el mensaje a todos los clientes registrados hasta el momento.
Nota: Como parte de la resolución debe definir el protocolo entre el `Timer` y sus clientes, por lo que se solicita que indique claramente el tipo de los mensajes intercambiados.

b) Un proceso `Cell` que toma por parámetro: (i) un canal por donde recibe el estado de sus vecinas; (ii) una lista con exactamente 8 canales por donde puede comunicar su estado a sus vecinas; (iii) un canal que identifica a un proceso `Timer`; y (iv) un booleano que indica si inicialmente está viva o muerta. La `Cell` debe registrarse al `Timer` y ante cada mensaje "tick" recibido debe enviar su estado a sus vecinas, recibir el estado de sus 8 vecinas, y finalmente actualizar su estado siguiendo las reglas enunciadas.