Universidad de los Andes
Ingeniería de Sistemas y Computación
ISIS 2203 Infraestructura Computacional

CASO I.

Manejo de la concurrencia

El sistema de gestión y rastreo de unidades de distribución *(*GDTS) recibe información de posición de las unidades de distribución (vehículos), y responde consultas sobre esta información.

En este caso queremos implementar el esquema de concurrencia para manejar las consultas que se hacen sobre el GDTS.

No se tomará en consideración la comunicación con los vehículos, solo la parte de las consultas.

En principio, las consultas consisten en un listado de vehículos de los cuales se desea saber la posición, sin embargo, en este caso, no tendremos en cuenta el contenido de esos mensajes, puesto que deseamos concentrarnos en el esquema de concurrencia. Para probar el programa, las consultas pueden consistir en números generados en secuencia, o al azar, y la respuesta en este número incrementado.

Objetivo

Diseñar un mecanismo de comunicación para manejar las consultas de múltiples clientes sobre el servidor GDTS. Para este caso, los clientes y el servidor serán *threads* en la misma máquina (en casos posteriores se implementará como un esquema distribuido; este es solo un prototipo).

El proyecto debe ser realizado en java, usando *threads*. Para la sincronización solo pueden usar directamente las funcionalidades básicas de Java: `synchronized`, `wait`, `notify`, `notifyAll, join y yield`.

Funcionamiento

Cada *thread* cliente hace un cierto número de consultas y termina. El número de clientes y el número de mensajes que envía cada uno deben ser un número arbitrario (cada cliente tendrá asignado un número particular de mensajes enviados). Para cada mensaje, el *thread* cliente debe generar un objeto de tipo Mensaje e inicializarlo, después lo envía. Cuando termine, el cliente le debe avisar al buffer que se retira.

El servidor, por su lado, estará compuesto por varios *threads* para poder atender múltiples consultas simultáneamente. Estos *threads* deben terminar cuando no haya más clientes. Los *threads* servidores estarán continuamente solicitando mensajes al buffer y respondiendo las respectivas consultas (responder consiste en incrementar el valor del mensaje y avisarle al cliente que puede continuar). El número de servidores también debe ser un número arbitrario.

El número de clientes, el número de servidores, el número de consultas de cada uno de los clientes y el tamaño del buffer deben estar en un archivo, el cual será procesado por el `main` del programa.

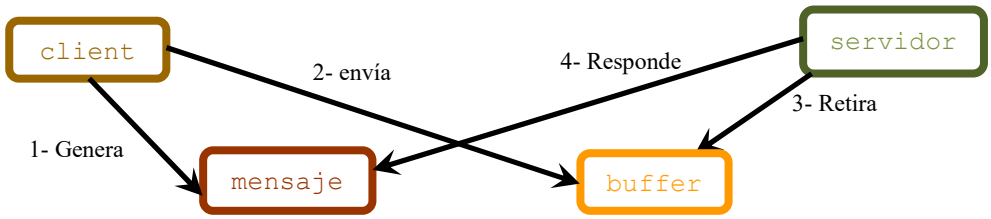
Diseño:

En el sistema tendremos: clientes, servidores, un buffer y mensajes. Los clientes y servidores son los antes descritos.

En cuanto al buffer, es un sitio donde los clientes almacenan los mensajes para que sean recogidos por los servidores; este buffer debe tener una cierta capacidad limitada, y funcionar en esquema productor-consumidor. Por su parte, los mensajes son objetos con la consulta que hace el cliente, y donde el servidor deja la respuesta.

El funcionamiento es el siguiente: un cliente genera un mensaje, e intenta depositarlo en el buffer; si no es posible, se duerme en el buffer (espera pasiva). Una vez depositado el mensaje, el cliente debe quedar a la espera de la respuesta del servidor; de nuevo espera pasiva en el mismo objeto buffer. Cuando detecta que su respuesta está lista termina con el mensaje actual y pasa al siguiente (o termina).

Cada servidor, por su parte, está continuamente intentando retirar mensajes del buffer; si no es posible, vuelve a intentarlo). Sin embargo, el servidor debe ceder el procesador después de cada intento (método `yield`, espera semi-activa). Una vez retirado el mensaje, genera una respuesta en el mismo mensaje del cliente, y procede a despertar a todos los clientes para que el interesado pueda detectar que su mensaje ha sido respondido. El esquema general es el siguiente:



Tanto el cliente como el servidor se comunican con el buffer; no se comunican directamente entre ellos. El buffer puede recibir la información de cuántos clientes hay, pero no de cuántos mensajes van a circular. El tamaño del buffer debe ser configurable; puede ser mayor, menor o igual al número de clientes. La sincronización implementada debe responder exactamente al comportamiento aquí descrito.

Condiciones de entrega

- En un archivo .zip entregar los fuentes del programa, y un documento word explicando el diseño (modelo conceptual del mundo incluido) y funcionamiento del programa. En particular, para cada pareja de objetos que interactúan, explique cómo se realiza la sincronización, así como el funcionamiento global del sistema. El nombre del archivo debe ser: caso1_*login1*_*login2*.zip
- El trabajo se realiza en grupos de **2** personas. No debe haber consultas entre grupos.
- El grupo responde solidariamente por el contenido de todo el trabajo, y lo elabora conjuntamente (no es trabajo en grupo repartirse puntos o trabajos diferentes).
- Se puede solicitar una sustentación a cualquier miembro del grupo sobre cualquier parte del trabajo. Dicha sustentación será parte de la calificación de todos los miembros.
- El proyecto debe ser entregado por Sicua+ por uno solo de los integrantes del grupo. **Al comienzo del documento Word, deben estar los nombres y carnés de los integrantes del grupo**. Si un integrante no aparece en el documento entregado, el grupo podrá informarlo posteriormente, sin embargo habrá una penalización: la calificación asignada será distribuida (dividida de forma equitativa) entre los integrantes del grupo.
- **Se debe entregar por Sicua+ a más tardar el viernes 11 de septiembre a las 23:55 p.m.**