**Infraestructura Computacional**

**Caso #1**

**Luis Miguel Gómez Londoño – 201729597**

**Daniel Andrés Bernal - 201519654**

**Diseño y Funcionamiento**

Se desarrolló el programa para que siguiera estrictamente los requerimientos estipulados en el documento del caso…

**Cliente**

El cliente en este caso posee un id, una referencia al buffer y además una lista de mensajes, como también un número de consultas (cuantos mensajes tiene). Lo único que hace el proceso del cliente es crear tantos mensajes como número de consultas y nada más… Por cada mensaje creado, se hace un intento de almacenamiento en el buffer…

**Mensaje**

El mensaje cuenta con un id, una referencia al buffer y al cliente a quien le pertenece… Además, contiene una variable entera de contenido inicializada en 0. Cuando el mensaje esta sin modificación alguna, esta es igual a 0. Cuando el mensaje ha sido modificado, su valor es modificado a 1 (a través del método *cambiar()*).

**Servidor**

El servidor cuenta únicamente con un id y con una referencia al buffer. El proceso del servidor es el siguiente: El servidor constantemente intenta hacer un retiro del buffer (con el método *buffer.retirar()*), luego se verifica que se haya podido hacer el retiro, y antes de hacerle cambio alguno, despierta los procesos que duermen dentro de él (ya que, si existe y se retiró, ya hay capacidad en el buffer disponible). Se realiza el cambio y se reinicia el proceso de retiro de mensajes. En caso que no hayan mensajes en el buffer (que esté vacío), se utiliza el método *yield()* para liberar el procesador que usan los threads de servidor. Si ya no hay mensajes restantes en el buffer para que procese, se sale del ciclo y el servidor muere.

**Buffer**

En el caso del buffer, se tienen las siguientes variables: *tamanio* indica el tamaño que va a tener el buffer, numClientes *y numServidores… consultasClientes* contiene el número de consultas o mensajes que va a crear y mandar cada uno de los clientes, *mensajesRestantes* hace el cálculo del total de mensajes que hay en el programa para atender (*numClientes \* consutlasClientes*). Obviamente el buffer cuenta con un *ArrayList* en donde almacena todos los mensajes. El buffer posee un método de leer archivo que permite seleccionar al usuario el archivo de configuración. Este archivo se debería de encontrar (aunque existe forma de seleccionarlo con interfaz) en la carpeta *data > config.txt.* La estructura de este archivo debe seguir el siguiente formato: *Numero de Clientes,Numero de Servidores,Numero Consultas Cliente, Tamanio Buffer*

Una vez seleccionado el archivo, se carga la configuración y variables en el programa… El buffer cuenta con un método *guardar(Mensaje mensaje)* que guarda el mensaje que le llegue por parámetro. Es el cliente quien llama a este método, por lo que varios synchronized que quieran tomar acción sobre el usuario se refieren a *this.* El método itera continuamente y por cada iteración verifica que el mensaje quepa en el buffer, si cabe, entonces se agrega el mensaje a la lista, se pone el mensaje en espera (todo lo que este dentro de él) y luego se sale de la iteración y se acaba el método. Si el mensaje no cabe en el buffer, se pone al cliente que está usando el método en espera. El beneficio de usar una iteración *while(true)* es que la única forma que se puede salir, es agregando un mensaje, por lo que cuando el cliente se despierte, se vuelve a verificar capacidad.

En el buffer también se encuentra el método que usan los servidores para retirar constantemente los mensajes. Este funciona así de simple: Primero se verifica que haya mensajes en la lista para poder retirar, liego se saca y se guarda en una variable. Una vez retirado de la lista se decrementa la variable que nos cuenta los mensajes restantes. Una vez retirado un mensaje se hace llamado al método *notifyAll()* para despertar cualquier proceso que esté esperando a una liberación del buffer. Este método trabaja de la mano con el método de guardar.

Por lo general en este programa se sincronizan bloques que va a usar el cliente en el buffer con *synchronized(this)* para que este pueda hacer acceso sincronizado. Se sincronizan mensajes sobre todo en el pedazo del código en donde se le aplican cambios, como por ejemplo guardar o retirar el mensaje directamente de la lista del buffer.