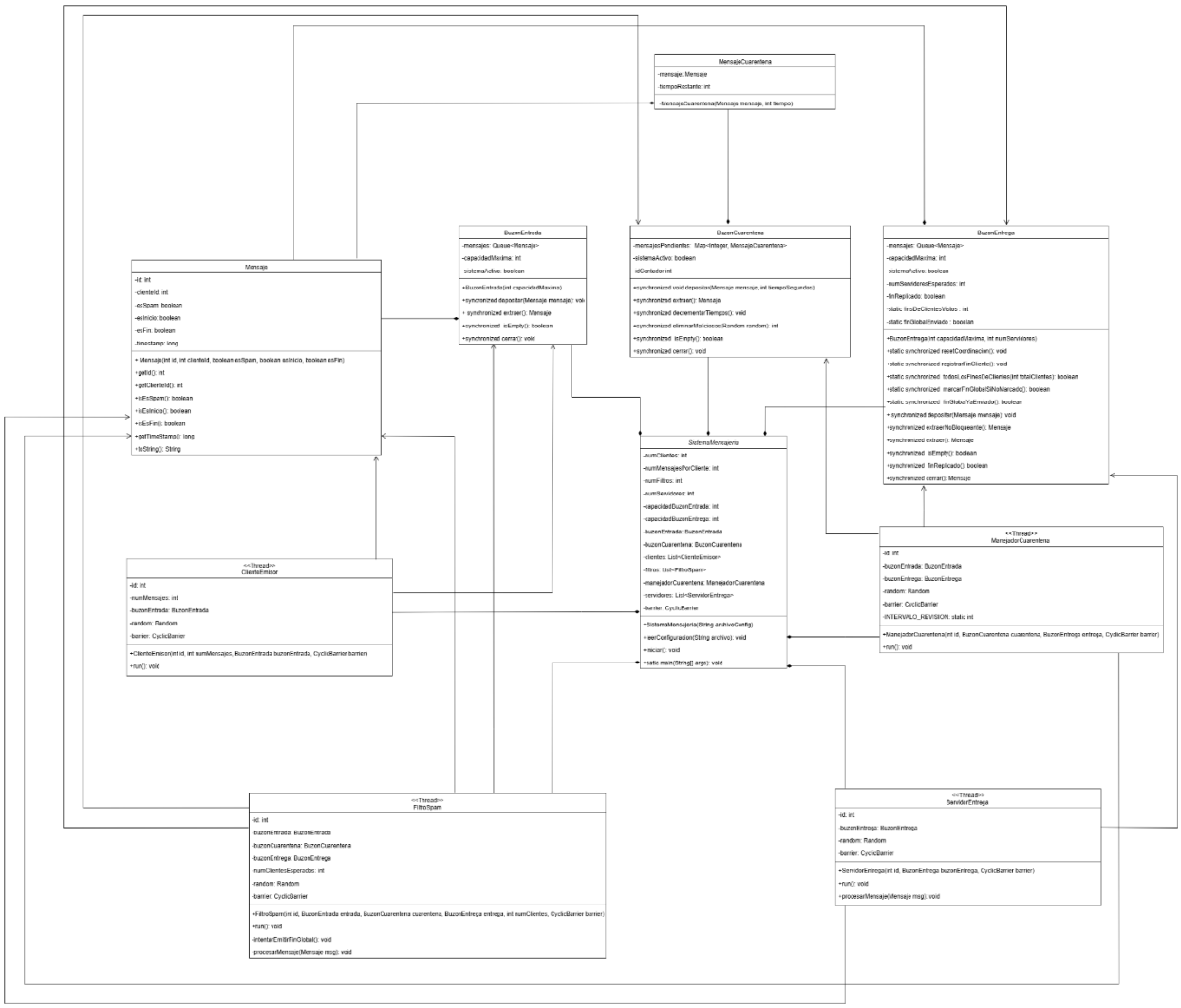
Caso 3 TIC

- Santiago Álvarez R – 202321575

-N Felipe Celis D - 202320636

1. Diseño del Sistema

Para este caso, se ha pedido modelar un sistema de mensajería en la nube. Este debe coordinar, entregar y los mensajes, garantizando el control de spam y disponibilidad del espacio. Para esto, el sistema fue diseñado siguiendo una arquitectura basada en productores y consumidores sincronizados, a través del uso de hilos y mecanismos que coordinan datos en Java. El modelo general está compuesto por tres tipos de entidades: los productores, que son simplemente las personas que escriben los mensajes, los procesadores intermedios encargados de filtrar el spam y manejar la cuenta, y finalmente los consumidores finales, los cuales son los servidores de entrega. Todos estos se comunican a través de tres buzones compartidos: BuzonEntrega, BuzonCuarentena y BuzonEntrada. El sistema inicia con la clase SistemaMensajeria, encargada de crear y coordinar todos los hilos y buzones.



2. Funcionamiento del Sistema

Para iniciar el sistema, nuestro SistemaMensajeria lee un archivo de configuración (config.txt) con los siguientes parámetros: número de clientes, filtros, servidores, y capacidades de buzones. Seguido de esto, se crean todos nuestros hilos y objetos compartidos, y estos últimos se asignan a los diferentes hilos. Para iniciar todos los hilos simultáneamente, se utiliza una barrera (CyclicBarrier).

Para la producción de mensajes, cada hilo de cliente debe mandar un mensaje inicial, seguido por unos mensajes normales, y de últimas un mensaje final. Estos mensajes son enviados y depositados en BuzónEntrada. Si el cliente quiere meter un mensaje al buzón, pero este está lleno, entonces hace una espera pasiva (wait.()) hasta que haya espacio.

Para el manejo de spam, nuestro filtroSpam consume los mensajes que llegan al BuzonEntrada, esperándolos en espera pasiva. Si un mensaje es spam, este es enviado al BuzonCuarentena, donde se hace un tiempo de retención aleatorio. Si este mensaje es válido, simplemente se envía al BuzonEntrega. Cada vez que se recibe un mensaje de fin, el filtro registra este evento en BuzonEntrega. Ya cuando todos los clientes hayan enviado su mensaje final y todos los buzones estén vacíos, un filtro genera el fin global del sistema.

El manejador de cuarentena es un hilo que reduce periódicamente el tiempo de los mensajes que están en cuarentena. Cuando un mensaje acaba su tiempo en esta zona, es extraído y enviado al BuzonEntrega. Adicionalmente, algunos mensajes son aleatoriamente catalogados como maliciosos. Este termina cuando recibe un mensaje final.

Ya para los servidores, que en otras palabras son los consumidores, esperan con espera activa a que los mensajes lleguen al BuzonEntrega. Cuando un servidor recibe un mensaje de fin, este acaba su ejecución. Al final, el BuzonEntrega informa del mensaje de fin a todos los servidores, asegurando que todos finalicen. Aquí, nuestro SistemaMensajeria espera con un join() a que todos los hilos terminen, y cierra todos los buzones. Finalmente, se imprimen todos los mensajes de confirmación de terminación de los múltiples componentes.

3. Sincronización entre pares de objetos

- ClienteEmisor-BuzonEntrada: Se utiliza una espera pasiva con syncronized, wait y NotifyAll. Aquí, el cliente deposita los mensajes en el buzón: si está lleno, se hace la espera hasta que haya cupo. Cuando se deposita un mensaje, se notifica a los filtros.
- FiltroSpam-BuzonEntrada: hace espera pasiva. Mientras el el buzón este vacío, FiltroSpam hace wait(). Cuando hay mensajes, otro hilo lo despierta con un NotifyAll().
- FiltroSpam-BuzonCuarentena: este aplica espera semiactiva. El filtro se encarga de depositar mensajes de spam en el buzón. Este no se bloquea: solo comunica y continua. El buzón lleva la cuenta del tiempo que un mensaje ha de esperar aquí.
- FiltroSpam-BuzonEntrega: utiliza espera semi-activa. Aquí los filtros depositan mensajes válidos que hayan pasado el filtro. Si esta lleno, se espera un tiempo y se vuelve a intentar.
- FiltroSpam-BuzonEntrega/Filtros: se utiliza variables estáticas y sincronización de la clase. Aquí, los métodos utilizan syncronized static para llevar la cuenta de mensajes de fin y coordinar el fin global. Solo un filtro puede generar el fin global: es único.
- ManejadorCuarentena-BuzonCuarentena: se hace una espera semiactiva periódica. Cada segundo se decrementa los tiempos de los mensajes de spam, se hacen eliminaciones de

mensajes maliciosos y revisa si hay mensajes listos. Para esta espera, no se utiliza un wait.() si no una dormida (Thread.sleep())/Thread.yield() para ceder procesador.

- ManejadorCuarentena-BuzonEntrega: están sincronizados son syncronized. Simplemente, cuando un mensaje está listo, es depositado en el buzón de entrega.
- ServidorEntrega-BuzónEntrega: utiliza una espera activa. Aquí, los servidores leen constantemente el buzón sin bloquearse: han de revisar constantemente el buzón para no bloquear el sistema de mensajeria. Para esto, si el buzón está vacío, simplemente se duerme brevemente (Thread.sleep(10)). Si hay mensajes, son procesados.
- CyclicBarrier-Todos los hilos: Sincronización inicial. Al principio, todos los hilos llaman a barrier.await() cuando están listos, y la barrera solo deja continuar si todos los hilos están listos. Esto garantiza que el sistema inicie en un estado coordinado.
- SistemaMensajeria-Todos los hilos: control de terminación. Aquí, el sistema principal espera a que todos los hilos terminen de trabajar con un join() para poder finalizar todos los recursos.

4. Validación y pruebas

Para validar el correcto funcionamiento del sistema de mensajería concurrente, se realizaron múltiples pruebas ejecutando el programa bajo distintas configuraciones, analizando tanto el comportamiento funcional como la correcta sincronización de hilos requerida en el enunciado.

Prueba inicial y logs de ejecución:

Para la prueba principal se ejecutó con el archivo config.txt de los siguientes parámetros:

CLIENTES=3

MENSAJES_POR_CLIENTE=10

FILTROS=2

SERVIDORES=2

CAPACIDAD_BUZON_ENTRADA=3

CAPACIDAD_BUZON_ENTREGA=5

Esta configuración permite evaluar el trabajo de 8 hilos simultáneamente (3 Clientes emisores, 2 filtros, 2 servidores de entrega y 1 manejador de cuarentena) sincronizados por medio del método CyclicBarrier.

Durante la ejecución los logs que se evidenciaron muestran la correcta cooperación y sincronización. Algunos Logs destacados que muestran la correcta ejecución son estos:

[Cliente-1] Iniciando envío de mensajes

[Filtro-0] INICIO de cliente 1

[BuzonEntrada] Cliente 1 depositó Msg[id=10001, cliente=1, tipo=VALIDO] (total: 3)

[BuzonEntrega] Depositado Msg[id=10001, cliente=1, tipo=VALIDO] (total: 4)

[Filtro-0] A entrega: Msg[id=10001, cliente=1, tipo=VALIDO]

[Servidor-0] Procesando Msg[id=10001, cliente=1, tipo=VALIDO]

[BuzonEntrada] Cliente 3 depositó Msg[id=30003, cliente=3, tipo=SPAM] (total: 1)

[Cuarentena] Depositado Msg[id=30003, cliente=3, tipo=SPAM] con tiempo=11s (total: 1)

[Filtro-0] A cuarentena: Msg[id=30003, cliente=3, tipo=SPAM] (tiempo=11s)

[Servidor-1] OK Msg[id=20001, cliente=2, tipo=VALIDO] (ciclos=559)

[Servidor-1] Procesando Msg[id=40001, cliente=4, tipo=VALIDO]

[Manejador-0] Eliminados 2 mensajes maliciosos

[Cuarentena] Extraído Msg[id=30004, cliente=3, tipo=SPAM] (listo para entrega)

[Manejador-0] Liberado de cuarentena: Msg[id=30004, cliente=3, tipo=SPAM]

[Manejador-0] FIN Entrega (Cuarentena vacía)

[BuzonEntrega] FIN replicado x2

[Servidor-0] Procesando Msg[id=30004, cliente=3, tipo=SPAM]

[Servidor-0] OK Msg[id=30004, cliente=3, tipo=SPAM] (ciclos=622)

[Servidor-1] FIN recibido. Cerrando.

[Servidor-0] FIN recibido. Cerrando.

[Servidor-1] Finalizó procesamiento

[Servidor-0] Finalizó procesamiento

Todos los servidores finalizaron

========== SISTEMA FINALIZADO EXITOSAMENTE ==========

Estos resultados nos permiten evidenciar que se generan correctamente los mensajes de inicio, validos, spam y fin, como los filtros de spam clasificaron los mensajes correctamente y enviándolos a sus buzones correspondientes, el manejador de cuarentena elimino y libero mensajes de acuerdo a los tiempos asignados, los servidores de entrega procesaron todos los mensajes válidos y se detuvieron al llegar el mensaje de fin, y el sistema imprimió todos los mensajes correspondientes a estas acciones de manera secuencia a como se ejecutaba el programa.

Validación sincronización:

-Para validar que la sincronización fuera correcta, se validó que la barrera funcionara de manera correcta, esperando que todos los hilos empezaran al mismo tiempo.

-Todos los buzones mostraron acceso concurrente y sin condiciones de carrera gracias al uso de diferentes métodos de sincronización como synchronized, wait y notify.

-Se verifico que no existieran ni se generaran bloqueos mutuos (deadlocks) y que todos los recursos compartidos se liberaron correctamente.

-Las esperas pasivas,semi-activas y activas se ejecutaron según los requerimientos del enunciado.

Pruebas adicionales:

Se observo el comportamiento de 3 casos diferentes; El caso base explicado anteriormente, Uno con una carga un poco más alta con 5 cliente, 3 filtros y 3 servidores, y finalmente uno con buzones de capacidad mínima. Todos estos casos se ejecutaron de manera correcta y confirma la correcta ejecución y funcionamiento del programa.

Conclusión validación:

El sistema fue validado y testeado con diferentes pruebas lo que permite evidenciar el correcto funcionamiento del programa bajo diferentes escenarios de prueba mostrando la correcta interacción entre entidades, la coordinación de los diferentes métodos de concurrencia según era requerido, la finalización y sincronización de todos los hilos en cada etapa del proceso y finalmente un manejo seguro de los recursos compartidos evitando los problemas de concurrencia (condiciones de carrera y bloqueos), llegando así a la conclusión que el programa cumple con los requerimientos necesarios y teniendo una estructura sólida.