UNIVERSIDAD NACIONAL DE CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales

Trabajo Práctico N° 1 - “Sistema de reservas de vuelos”

Programación Concurrente

Grupo: *Threading Bad*

- PALACIO, LUIS ENRIQUE *(40.609.862)*
- RODRIGUEZ, MATEO *(43.967.398)*
- SAILLEN, SIMON *(42.978.650)*
- TRACHTA, AGUSTIN (*43.271.890*)
- VARGAS, RODRIGO SEBASTIÁN *(42.016.802)*

DESARROLLO:

I) Trabajo Realizado

Descripción de las clases

A partir del enunciado del problema, se planteó el programa con 8 clases:

1. **Main**: Es en donde se inicializa el programa. Se instancia la clase *Registros* y esta se pasa como argumento a las clases que representan las diferentes Etapas.
   También se instancia una clase Estadística que implementa la interfaz *Runnable* y esta se ejecutará mediante el hilo *statThread*.
   Luego, se dará inicio a los diferentes hilos, esto se hace mediante los métodos *ejecutarEtapa()* disponibles en cada una de las "clases etapa", y en caso del *statThread* se realiza directamente en *main*.
   Por último, se espera a la finalización del hilo *statThread* asociado a la clase *Estadística* para posteriormente imprimir el tiempo que pasó entre la primera instancia de *Registros* y la finalización del programa cuando los diferentes hilos terminan
2. **EtapaReserva:** Dentro de esta clase se encuentra la clase *ThreadReserva* implementada *Runnable*. En el método *run()* se lleva a cabo la reservación de un asiento aleatorio de la matriz de asientos (buscando otro nuevo aleatorio en caso de no encontrar asiento disponible). Esta etapa es llevada a cabo por 3 hilos, inicializados mediante el método *ejecutarEtapa()*. Los mismos tienen su propio tiempo de acción por iteración y los hilos morirán al no tener asientos disponibles.
3. **EtapaPago:** En esta clase se encuentra la clase privada *ThreadPago* implementada *Runnable*. En el método *run()* se lleva a cabo el pago de los asientos, se elige un asiento al azar de la lista de asientos pendientes, aceptando el pago con una probabilidad del 90% confirmando la reserva. Esta etapa es llevada a cabo por 2 hilos, inicializados mediante el método *ejecutarEtapa()*. Los mismos tienen su propio tiempo de acción por iteración, estos dormirán un tiempo determinado, comprobarán si hay reservas pendientes y en caso de no encontrar morirán.
4. **EtapaValidación:** Dentro de esta clase se encuentra la clase privada *ThreadValid* implementada *Runnable*. En el método *run()* se lleva a cabo la validación de las reservas pagadas, se elige una reserva aleatoria dentro de la lista y con una probabilidad del 10% se cancela dicha reserva, quedando el resto con la marca *Checked* pasando a la siguiente etapa. Esta etapa es ejecutada por 3 hilos, los

mismos tienen su determinado tiempo de acción por iteración, estos hilos dormirán y despertarán intermitentemente debido a que el flujo de reservas a validar no es constante, y una vez que no queden más morirán.

5. **EtapaVerificación:** En esta clase se encuentra la clase privada *ThreadVerif* implementada Runnable. Dentro de la misma se encuentra el método *run()*, donde se llevará a cabo la verificación de las reservas aleatoriamente, guardando en una lista las reservas que se encuentran con la marca *Checked* y descartando las otras. Esta etapa es ejecutada por 2 hilos con su tiempo de acción por iteración determinado, los mismos se dormirán intermitentemente debido a que el flujo de asientos a verificar no es constante, y morirán al no encontrar más.
6. **Registros:** Esta clase es la más importante, en ella se encuentran los registros de todo, la matriz de asientos, las listas de reservas pendientes, confirmadas, verificadas y canceladas, también se encuentran los métodos synchronized, los cuales nos permiten evitar problemas de datos en el manejo de las reservas
7. **Asiento:** Esta clase fue creada para almacenar los datos y estados de los asientos del vuelo, posee pocos métodos (sobretodo *setters* y *getters*) pero es altamente utilizada por la clase *Registros*.
8. **Estadística:** Esta clase implementa la interfaz Runnable, la misma se ejecutará con un solo hilo. Su objetivo es escribir en un archivo de texto cada 200 [ms] las estadísticas de los registros de reservas al momento, hasta el final del programa.

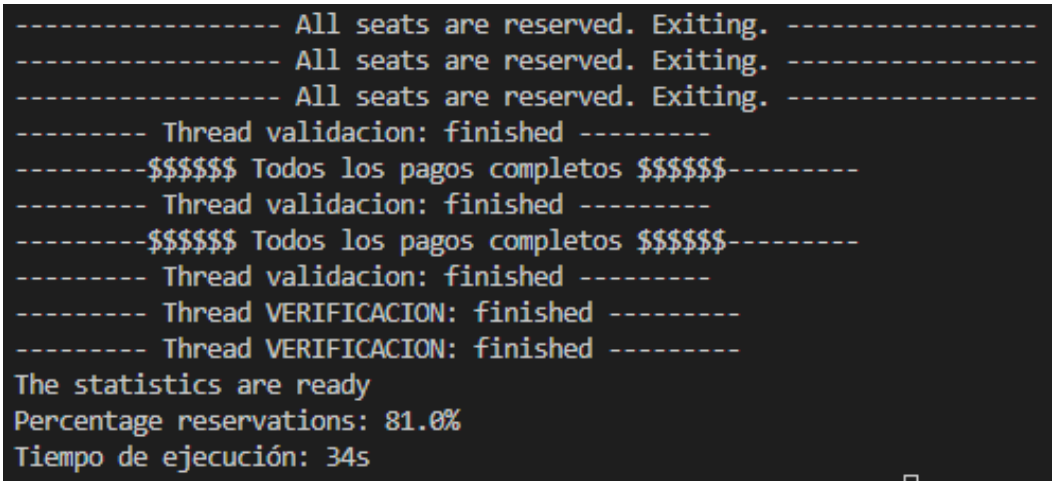
Dinámica de ejecución

Los asientos se crean e inicializan en Registros en forma de matriz. A partir de esta matriz, comienza todo el sistema de reserva de vuelos. La primera etapa, *Etapa Reserva*, selecciona aleatoriamente un asiento de la matriz e intenta reservarlo. Una vez reservado, el asiento pasa a un estado pendiente (indicando que se espera el pago). Luego, en la *Etapa Pago* se intenta confirmar o cancelar la reserva con una probabilidad del 90%.

Posteriormente, basándose en las confirmaciones de pago, la *Etapa Validación* intenta validar la reserva. Al igual que la *Etapa Pago*, esta etapa tiene una probabilidad del 90% de validar la reserva o, en su defecto, cancelarla. Finalmente, la última etapa, *Etapa Verificación*, se encarga de revisar las reservas que han sido confirmadas y validadas.

A pesar de explicarse de manera secuencial, todas las etapas comienzan al mismo tiempo y funcionan de manera concurrente.

Impresiones del log y terminal



Tests y Pruebas de tiempo de iteración:

Para el propósito de estas pruebas, el tiempo de sleep será fijado e igual para todos los hilos (exceptuando Reserva) [1 segundo].

TEST BASE:
Reserva: 400 - Pagos: 300 - Validaciones: 200 - Verificaciones: 150

- Max Cola Pendientes: 27
- Max Cola Confirmadas: 7
- Tiempo: 29-30 [s]
- Errores: 1 vez los hilos Pago no murieron. (posiblemente se quedaron esperando el notifyAll).

Observaciones: Los hilos Reserva mueren antes que el resto, aproximadamente a 152-158

asientos procesados dejando 24-26 reservas pendientes, hilos Verificacion y Validacion mueren casi al unísono después de Pago. En algunos intentos Verificación terminó antes que Pago.
Conclusiones: Tiempo de Reserva muy rápido, tiempo de Pagos muy rápido.

TEST 1: (Aumento Tiempo de Reserva)
Reserva: 600 - Pagos: 300 - Validaciones: 200 - Verificaciones: 150

- Max Cola Pendientes: 6
- Max Cola Confirmadas: 6
- Tiempo: 38-42 [s]
- Errores: 1 vez un hilo Verificación murió antes que todos.

Observaciones: Los hilos Reserva ahora mueren cerca del resto, aproximadamente a 162-185, el resto de los hilos termina en orden. Muchas veces quedan vacías las listas, retrasando al resto de los hilos. 1 vez un hilo Pago murió al último. 1 vez los hilos Verificación murieron antes que Validación. 2 hilos de Validación murieron antes que Pago.
Conclusiones: Tiempo de Reserva mejor, aunque un poco más lento del ideal, valor recomendado 500.

TEST 2: (Disminución Tiempo de Pago, Disminución Tiempo Reserva)
Reserva: 500 - Pago: 250 - Validaciones: 200 - Verificaciones: 150

- Max Cola Pendientes: 7
- Max Cola Confirmadas: 11 (32 en 1 escenario)
- Tiempo: 32-37 [s]
- Errores: 1 vez un hilo Validación murió antes que todos.

Observaciones: Los hilos Reserva terminan cerca del resto, aproximadamente a 157-182, el resto de los hilos termina en orden.
Conclusiones: Los tiempos de reserva y pago parecen estar acordes, hay pocos casos donde las últimas etapas sufren problemas.

TEST 3: (Aumento tiempo Validaciones)
Reserva: 550 - Pago: 250 - Validaciones: 300 - Verificaciones: 150

- Max Cola Pendientes: 7
- Max Cola Confirmadas: 43
- Tiempo: 37-44 [s]
- Errores: Los hilos de Verificación suelen morir antes de tiempo.

Observaciones: En casi todos los escenarios, mueren uno o ambos hilos de Verificación, retrasando o rompiendo la ejecución.
Conclusiones: Los tiempos de reserva y pago parecen estar acordes, el tiempo de la etapa de validación parecería estar bien, el tiempo de verificación es muy bajo.

TEST 4: (Aumento tiempo Verificación)
Reserva: 550 - Pago: 250 - Validaciones: 300 - Verificaciones: 300

- Max Cola Pendientes: 7
- Max Cola Confirmadas: 41 (+94 en varios escenarios)
- Tiempo: 45-70[s]
- Errores: En 1 escenario dos hilos de validación murieron antes. En 1 escenario, uno de verificación.

Observaciones: Los tiempos son muy altos debido a que los hilos suelen morir antes de tiempo.
Conclusiones: Esta configuración no resultó. La etapa reserva funciona correctamente, el resto de los hilos parecen tener problemas de tiempo. Valores propuestos: Pago 250, Validaciones 250, Verificaciones 200.

TEST 5: (Disminución tiempo Validación, Disminución tiempo Verificación).
Reserva: 550 - Pago: 250 - Validaciones: 250 - Verificaciones: 200

- Max Cola Pendientes: 6
- Max Cola Confirmadas: 12 (50)
- Tiempo: 37-52 [s]
- Errores: Suele morir un hilo de Verificación antes de tiempo.

Observaciones: En los casos donde los hilos no mueren antes de tiempo los tiempos parecen estar acordes.
Conclusiones: Estos serán los tiempos propuestos. Aunque, después analizar, optamos por bajar a 100 el tiempo de Verificaciones, esto es debido a que cuando la lista de Confirmadas no es 0 los hilos recorren todo el tiempo en búsqueda de la flag Checked, y si aumentamos mucho el tiempo de iteración afecta el rendimiento general de todo el código.

Tiempos de Iteración Propuestos:
Reserva: 550 - Pago: 250 - Validaciones: 250 - Verificaciones: 100

A parte de estos tiempos, los tiempos de dormido serán generados aleatoriamente entre un rango determinado, y en el caso de la etapa Verificación decidimos que el rango de dormido sea mayor, debido a que es el hilo que puede generar más problemas si se muere antes de tiempo (como vimos en los test).

II) Decisiones de diseño tomadas y su justificación

Mecanismos de control de la finalización del programa

Para cada uno de los Threads correspondientes a cada etapa, hay un punto de quiebre específico:

Etapa Reserva: Sus threads terminarán cuando se haya recorrido la matriz por completo, es decir, cuando todos los asientos hayan sido reservados al menos hasta un estado de pendiente.

Etapa Pago: Una vez que las reservas pendientes cesen, es decir, todas las reservas hayan pasado por una confirmación de pago o una cancelación y un tiempo de espera propio, los threads de esta etapa finalizarán.

Etapa Validación: Una vez que no haya más reservas confirmadas, es decir, todas las reservas confirmadas ya fueron chequeadas o canceladas, el hilo morirá luego de esperar un tiempo propio y no encontrar más reservas.

Etapa Verificación: Luego de observar que no haya más reservas confirmadas con la flag *Checked* o cuando no queden más reservas confirmadas, el hilo morirá luego de esperar un tiempo propio y no encontrar más reservas.

Hilo Estadística: Este hilo llegará a su fin cuando la suma de reservas canceladas y verificadas sea igual al máximo de asientos. En este momento, será recibido por el *join* del *main* y se terminará la ejecución de todo el programa. Los hilos irán muriendo casi secuencialmente por etapa (dependiendo del tiempo de sleep de cada uno) hasta el último hilo, que será el de Estadística. El main esperará a este hilo y, al final, imprimirá lo que estuvo en ejecución durante todo el programa.

Mecanismos de control de la concurrencia de los hilos

Los bloques synchronized fueron utilizados en la mayoría de los métodos de Registros (en aquellos donde se hace algo alrededor de las listas) en donde el lock de cada uno de los bloques son las diferentes listas (excepto en *get_asiento()* que el lock es la matriz). A su vez, dentro del método *run()* de cada uno de los hilos (excepto por Estadística) los bloques synchronized del código se sincronizan con el asiento a tratar entre los diferentes tipos de reserva, ya que no se quiere que haya varios hilos interactuando con el mismo asiento al mismo tiempo.

Además, utilizamos clases atómicas, como *AtomicInteger*, para el conteo de asientos disponibles/libres dentro de la clase Etapa Reserva. Esto se debió a que en algunas instancias el contador podía corromperse al cambiar el valor entre hilos. La solución con AtomicInteger garantiza la consistencia en este tipo de operaciones concurrentes.

III) Las conclusiones obtenidas en base a los resultados

Luego de múltiples ejecuciones del programa, se pudo observar que si se modifican los tiempos de duración por iteración de alguna etapa, esto puede afectar no solo la duración final del programa, sino también el comportamiento y resultados esperados, especialmente si una etapa depende del estado en que quedaron las reservas de otra etapa.

Por ejemplo, si se aumenta significativamente el tiempo de duración de la Etapa Reserva, es posible que los hilos de las etapas posteriores (como Pago, Validación y Verificación) finalicen antes de que se realice la reserva de todos los asientos. Esto podría llevar a resultados inesperados o a un funcionamiento incorrecto del programa.

Por otro lado, si se disminuye demasiado el tiempo de duración de alguna etapa, es posible que los hilos de esa etapa no tengan suficiente tiempo para completar su trabajo correctamente, lo que también podría afectar el resultado final del programa.

Por lo tanto, es importante tener en cuenta las dependencias entre las etapas y ajustar los tiempos de duración de manera cuidadosa para garantizar un funcionamiento correcto y coherente del programa.