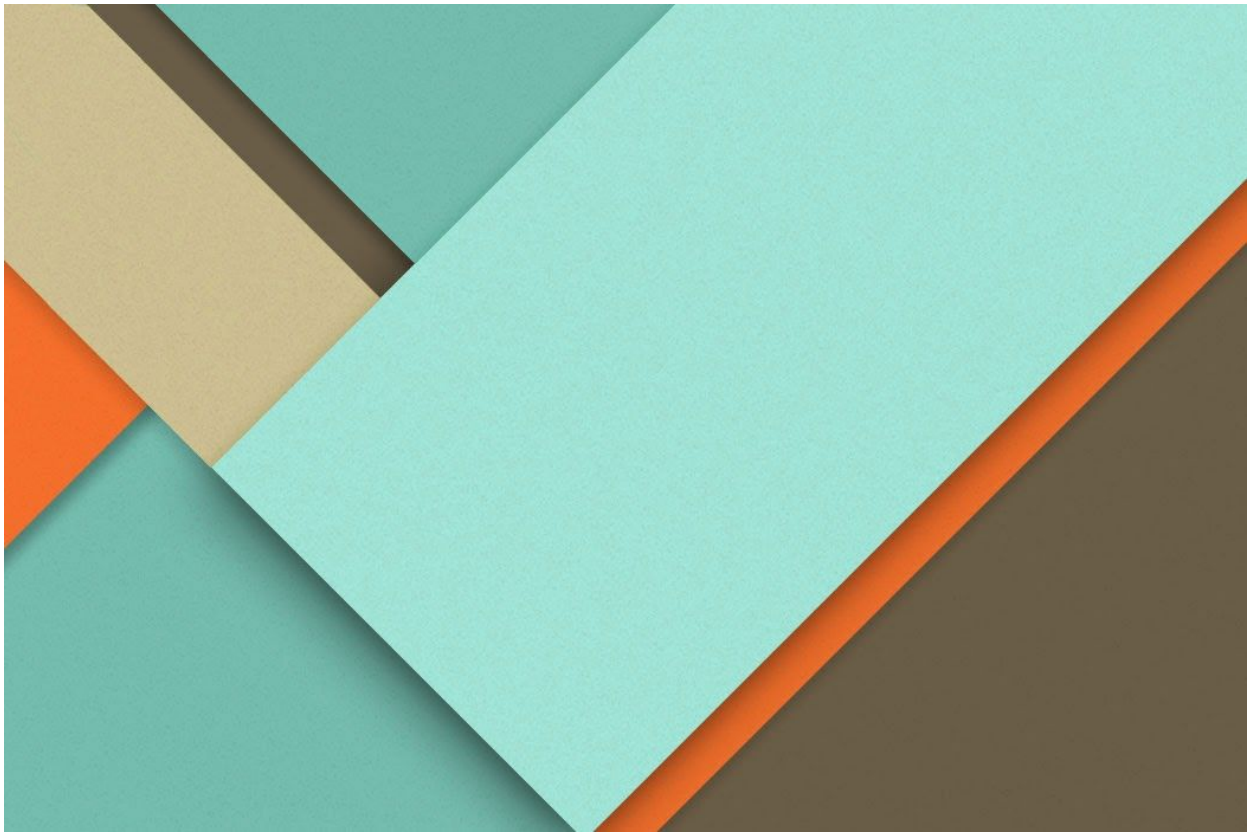


Manual Técnico

05.10.2020

Rubén Emilio Osorio Sotorro	201403703
Yoselin Annelice Lemus López	201403819
Brandon Bredly Alvarez López	201403862

Índice

Para cada problema se deberán identificar las situaciones en las cuáles existen múltiples procesos, las variables o datos que comparten los procesos, la forma en que se sincronizan y las situaciones que pueden llevar a problemas como : condiciones de carrera, deadlocks, livelocks ó inconsistencia de datos. Además se debe indicar cómo se solucionan dichos problemas y qué herramientas y conceptos se implementaron, todo esto deberá ir en la documentación del proyecto.

- Partes del programa en donde existieron múltiples procesos trabajando de forma concurrente y/o paralela.
- Cómo se realizó la comunicación y sincronización entre procesos.
- Situaciones en las cuáles era posible que se dieran: deadlocks, livelocks, condiciones de carrera, etc y cómo se solucionaron.
- Variables o datos que era necesario compartir entre procesos.

Problema 1: centro de acopio

Múltiples procesos que trabajan de forma concurrente y/o paralela

- Llegada de las personas a ingresar y extraer cajas de estantería
- Ver cola de extracción o ingreso de la estantería.

Comunicación y sincronización entre procesos

Sincronización

Al momento de ingresar caja en estantería y esta se encuentra vacía o el espacio está ocupado, se manda a la cola a esperar. Del mismo modo se da cuando se quiere retirar alguna caja, si está vacía o el índice a sacar no hay nada, éste se manda a cola a esperar.

Comunicación

Se utiliza una linkedlist para almacenar los índices en donde se quería insertar en la estantería, y otra para almacenar los índices en donde se quería retirar la respectiva caja. ejecutándose por medio de un hilo respectivamente.

Problemas

El manejo de inserción y extracción de cajas de la estantería ya que puede que se intente sacar una caja con más de 1 hilo a la vez, de la misma manera ingresar una caja a la estantería.

Solución

La solución es por medio de los bloqueos, para asegurar que cada acción se termine y no sea interrumpida por otro proceso.

Se hizo uso de las herramientas:

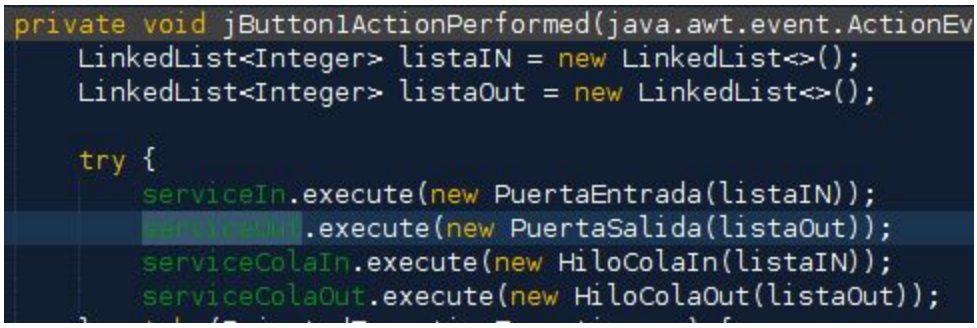
- newCachedThreadPool
- ReentrantLock

Variables o datos que era necesario compartir entre procesos

Para este primer problema se tienen cuatro procesos, el primero consta de la llegada de las personas a dejar caja, la llegada de las personas a recoger cajas, la verificación de la cola de personas esperando para dejar cajas, y la verificación de la cola de personas para sacar cajas.

Para generar la situación del problema es necesario compartir información para lo cual se compartió lo siguiente:

- La variable llamada *listaIn* es la compartida por los procesos, la cual es pasada como parámetro.
- La variable llamada listaOut es la compartida por los procesos de salida, la cual es pasada como parametro



Bloqueos

El manejo de los bloqueos se realizó sobre las clases *PuertaEntrada, PuertaSalida, HiloColaIn, e HiloColaOut* en el cual se encontraban las operaciones de colocar caja al llegar y desde cola, así como remover caja de la estantería desde la cola y desde la llegada.

Para los bloqueos fue utilizado *ReentranLock*, el cual cuenta con los metodos “lock y unlock”, los cuales bloquean y desbloquean respectivamente los procesos.

```
try {
    this.lockIn.lock();
    int indice = Estanteria.getIndiceRandom();
    if(Estanteria.espaciosOcupados < 20){
        Ventana.logs.append("Se quiere colocar caja en indice " + indice + ", espacios ocupados: " + Estanteria.espaciosOcupados + "\n");
        if(Estanteria.verDisponibilidad(indice) == 1){
            Ventana.logs.append("Quiso colocar en indice ocupado: " + indice + " , va a cola\n\n");
            this.listaIn.add(indice);
            Ventana.lblTotalColaIn.setText(String.valueOf(this.listaIn.size()));
            return;
        }
        Estanteria.llenandoIndiceEstanteria(indice);
        Ventana.arregloLabel[indice].setBackground(Color.BLACK);
        Ventana.logs.append("Se colocó caja en indice " + indice + "\n\n");
    }
    else{
        Ventana.logs.append("Error al ingresar caja en indice" + indice + ", la estanteria está llena, mandando a cola\n\n");
        this.listaIn.add(indice);
        Ventana.lblTotalColaIn.setText(String.valueOf(this.listaIn.size()));
    }
} catch (Exception e) {
} finally {
    this.lockIn.unlock();
}
```

Problema 2: el barbero dormilón

Múltiples procesos que trabajan de forma concurrente y/o paralela

- Llegada de los clientes
- Atención de los clientes por el barbero mientras otros llegan a la barbería

Comunicación y sincronización entre procesos

Sincronización

Este punto se da cuando el barbero está en dormido y llega un cliente, pero cuando este está atendiendo a su respectivo cliente, otro llega a sentarse a la cola de la barbería. En este punto debe esperar que termine con el cliente actual antes de ser atendido. En este punto es lo que sucede con la sincronización de los procesos en el cual hay ciertos puntos donde un proceso tiene que esperar que el otro termine para continuar.

Comunicación

Esto se produce en que el hilo que lleva la administración de las llegadas de los clientes y el del barbero tiene un objetivo en común. Desde la perspectiva del cliente es ser atendido por el barbero, en cuanto al punto de vista del barbero es atender a sus clientes.

Por lo cual los procesos requieren compartir recursos... no solo requieren los recursos como la memoria o el cpu en este caso también comparten la información que esta almacena en la estructura del linkedList utilizada en esta ocasión para el manejo de los clientes del problema. Esta estructura sirve para almacenar la llegada de los nuevos clientes además de que es el lugar al cual el barbero recogerá a un cliente para que este sea atendido, para lo cual también necesita acceder a la información almacenada en dicha estructura.

Problemas

La llegada concurrente de clientes a la barbería causa problemas dado que puede que se sobreescribieran los puesto de los clientes ya ocupados y de los que llegaban causando que no se llevara bien el control de los clientes que estaban en la cola y lo que se fueron.

Solución

La solución es por medio de los bloqueos, para asegurar que cada acción se termine y no sea interrumpida por otro proceso.

Se hizo uso de las herramientas:

- newCachedThreadPool
- ReentrantLock
- newCondition

Variables o datos que era necesario compartir entre procesos

Para el problema del barbero se tiene dos procesos, el primero consta de la llegada de los clientes a la barbería y el otro proceso consiste en la atención de los clientes por parte del barbero.

Para generar la situación del problema es necesario compartir información para lo cual se compartió lo siguiente:

- la variable llamada *cola* es la compartidas por los procesos, la cual es pasada como parámetro.

Aspectos del programa

Manejo de concurrencia

En este caso para manejar la concurrencia de la llegada de los clientes se hizo uso de *newCachedThreadPool*, el cual permite crear una piscina de hilos como su nombre lo indica, permitiendo la simulación de la llegada concurrente de los clientes.

```
private void btn_iniciarActionPerformed(java.awt.event.ActionEvent evt) {
    // Creando pool de hilos para llegadas concurrentes
    ExecutorService ejecutor = Executors.newCachedThreadPool();

    int clientes_entrada = Integer.parseInt(jtf_numero_clientes.getText());
    ejecutor.execute(new HiloBarbero(cola, clientes_entrada));
    ejecutor.execute(new Hilo(cola, clientes_entrada));

    ejecutor.shutdown();
}
```

Bloqueos

El manejo de los bloqueos se realizó sobre la clase *ColaSilla* en el cual se encontraban las operacion de insertar y remover de la cola a los clientes. Para fines de que estos procesos fuera exitoso para cada uno de los clientes sin la interferencia de otros procesos, fue necesario el uso de los bloques.

Para los bloqueos fue utilizado *ReentranLock*, el cual cuenta con los metodos “lock y unlock”, los cuales bloquean y desbloquean respectivamente los procesos.

Además, se hizo uso de *newCondition*, el cual posee los métodos de “wait y signalAll”, los cuales hace que un proceso se quede en espera o libera a un proceso de todos los wait para que este continúe con su flujo normal.

```
public void insertar() {
    bloqueo.lock();
    try {
        while (this.getContador() == this.getMax()) {
            noLleno.wait();
        }

        //Agregando el elemento
        for (int i = 0; i < cola.size(); i++) {
            Silla e = (Silla) cola.get(i);
            if (!e.isEstado()) {
                e.setEstado(true);
                e.setColor(e.getElemento(), Color.red);
                this.contador++;
                break;
            }
        }
        noVacio.signalAll();
    } catch (Exception e) {
    } finally {
        bloqueo.unlock();
    }
}
```

Problema 3: video juego space invaders

Múltiples procesos que trabajan de forma concurrente y/o paralela

- Múltiples enemigos
- Dos jugadores en una misma pantalla.
- Múltiples disparos de los jugadores.
- Múltiples disparos de los enemigos.
- Detección de colisiones.

Comunicación y sincronización entre procesos

Sincronización

Este punto se da cuando el disparo del enemigo pega con el jugador o el disparo del jugador choca con enemigo

Comunicación

Se utilizan cuadros imaginarios o rectángulos delimitadores alrededor de cada objeto, como un casco de colisión, y luego determinar si los cascos de colisión se superponen.

Problemas

Los controles cruzados pueden causar problemas de sincronización en los componentes, ya que estos controles no son seguros para subprocesos. Esto significa que varios subprocesos pueden acceder a un control al mismo tiempo, lo que puede provocar errores, por ejemplo, en el valor numérico del control, como el control de texto de la puntuación del jugador, si se leen y escriben varios hilos al mismo tiempo. Por ejemplo, un hilo puede ser en la mitad de la escritura de un valor mientras otro hilo lee el valor, lo que lleva a que se lea un valor incorrecto. En este programa en particular.

Solución

En lugar de crear subprocesos, se recomienda que se realicen tareas simples de subprocesos múltiples, usando un threadpool, por cada grupo de enemigos y por elementos que acompañan la pantalla.

Se hizo uso de las herramientas:

- newCachedThreadPool

Variables o datos que era necesario compartir entre procesos

La posición tanto de los enemigos conforme iba transcurriendo el tiempo con la trayectoria de la bala tanto enemiga como del jugador, para saber si había colisionado no o no.

También la posición de los enemigos para saber si ya habían llegado al fondo de la pantalla lo cual denotaba pérdida para los jugadores.

La posición del jugador 1 y el jugador 2 para no permitir que ambos se traslaparan ni colisionaran.