Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Sistemas Operativos 2
2do Semestre 2021
Ing. César Batz Saquimux
Aux. Brayan Chinchilla

Proyecto
Programación Concurrente

Objetivos

- Poner en práctica los conocimientos adquiridos sobre procesos e hilos.
- Resolver problemas que requieren técnicas de programación concurrente.
- Desarrollar software utilizando de forma eficiente múltiples hilos de ejecución y los recursos del sistema operativo.
- Coordinar múltiples procesos y tareas en programas utilizando los conocimientos adquiridos sobre programación concurrente para la correcta comunicación y sincronización entre procesos.

Instrucciones Generales Para Todos Los Problemas

En los grupos establecidos del laboratorio, se les solicita a los estudiantes desarrollar las siguientes aplicaciones y resolver los siguientes problemas utilizando el lenguaje de programación Java; y, sobre todo, aplicando de forma correcta los conocimientos vistos en el laboratorio sobre programación concurrente y desarrollo de software con múltiples hilos. Deberá analizar cada uno de los problemas y desarrollar los algoritmos necesarios para su solución.

Todos los problemas deberán contar con una interfaz gráfica o forma de visualizar el comportamiento de los mismos y cómo las operaciones con múltiples procesos influyen en ellos. Solamente se podrán utilizar las herramientas vistas en el laboratorio tales como: Thread Pools, Synchronized, ReentrantLock, etc. No se podrán utilizar otro tipo de estructuras concurrentes *built-in* en el lenguaje, se deberán desarrollar las propias. El uso de cualquier otra herramienta no vista en el laboratorio deberá ser consultado previo a su implementación para asegurar que se sigan cumpliendo los objetivos de la practica.

Para cada problema se debe realizar un análisis de la situación, en el análisis 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 o 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 y será de gran valor e importancia, ya que es una forma de evaluar que el estudiante tiene conocimiento del software que implementó. En concreto, se evaluará que cada uno de los manuales técnicos describan:

- Uso de hilos
- Recursos compartidos
- Comunicación y sincronización entre procesos.
- Situaciones en las cuáles era posible que se dieran: deadlocks, livelocks y cómo se solucionaron.

Con el objetivo de visualizar los cambios, será necesario utilizar el método Thread.sleep(n) para realizar una pausa y que los cambios sean apreciables. En los lugares en donde se aplique dicho método el valor de “n” que representa milisegundos debe ser aleatorio y el numero de milisegundos máximo debe ser modificable en medio de la ejecución a través de la interfaz grafica, debido a que de lo contrario no será apreciable la operación de múltiples procesos de forma concurrente y/o paralela.

Además, con el fin de evaluar la consistencia del sistema en un punto fijo en el tiempo, cada simulación deberá tener la opción de pausarse y resumirse en cualquier momento.

Problema 1: Centro de Acopio

Se tiene un centro en el cual se reciben y se entregan cajas con productos, el centro tiene una estantería con una capacidad máxima de 20 espacios. Existen dos puertas grandes: una para las personas que llegan a dejar su respectiva caja (cada persona lleva 1 caja) y la otra para las que llegan a retirar (cada persona puede retirar 1 caja).

Las cajas pueden ser pequeñas o grandes. Las pequeñas ocupan 1 espacio, y las grandes ocupan 2 espacios de la estantería. No existe una distinción de prioridad entre los tipos de cajas y su derecho a la estantería.

Múltiples personas pueden llegar al mismo tiempo al centro de acopio y pueden simultáneamente colocar cada una de ellas su caja en los lugares vacíos de la estantería, si la estantería está llena no pueden entregar sus cajas y deben esperar a que lleguen personas a recoger para que existan espacios vacíos para colocar la caja que llevan. De una forma similar, múltiples personas pueden llegar al centro y simultáneamente retirar cada una de ellas una caja de la estantería, si la estantería está vacía deben esperar a que lleguen personas a dejar cajas para entonces retirar.

Se debe modelar y desarrollar un sistema capaz de representar este comportamiento con las restricciones del negocio que sean obvias y lógicas, algunos ejemplos de estas son:

- Múltiples personas no pueden colocar su caja en el mismo espacio de la estantería.
- Múltiples personas no pueden retirar la misma caja de la estantería.
- Las personas deben colocar su caja en cualquier espacio disponible, no esperar a que un lugar especifico de la estantería se desocupe.
- Las personas deben retirar una caja de cualquier espacio disponible, no esperar a que un lugar especifico reciba una caja.

Además de modelar todas las interacciones, la aplicación deberá mostrar:

- Numero de espacios ocupados actualmente.
- Numero de cajas pequeñas colocadas.
- Numero de cajas grandes colocadas.
- Numero de cajas pequeñas retiradas.
- Numero de cajas grandes retiradas.

Parámetros modificables durante ejecución:

- Frecuencia de llegadas de productores de cajas pequeñas.
- Frecuencia de llegadas de productores de cajas grandes.
- Frecuencia de llegadas de consumidores de cajas pequeñas.
- Frecuencia de llegadas de consumidores de cajas grandes.

Problema 2: El Barbero Dormilón

El siguiente problema fue formulado por Edsger Dijkstra.

Existe una barbería en donde el barbero que la atiende corta el cabello a los clientes que llegan y cuando no hay ninguno, se pone a dormir. El barbero tiene una silla para cortar el cabello a donde atiende a un cliente y una sala de espera con 20 sillas en donde pueden sentarse los clientes que llegan mientras esperan. Cuando el barbero termina de cortar el cabello a un cliente, regresa a la sala de espera a ver si hay personas esperando, si las hay trae consigo a una persona para cortarle el cabello. Si no hay clientes esperando, se pone a dormir en la silla para cortar cabello.

Cada cliente que llega a la barbería observa lo que el barbero está haciendo. Si el barbero se encuentra durmiendo, el cliente lo despierta y se sienta en la silla para cortar el cabello. Si el barbero está cortando el pelo a alguien, entonces el cliente se coloca en una silla de la sala de espera. Si no hay sillas disponibles, entonces el cliente se va del lugar.

Se deberá desarrollar el software que modele el problema e implementar los algoritmos necesarios para solucionar los posibles problemas que puedan existir al ejecutarse múltiples procesos.

Además de modelar todas las interacciones, la aplicación deberá mostrar:

- Conteo de clientes que han visitado la barbería.
- Conteo de clientes que visitaron la barbería y se fueron sin esperar.
- Conteo de clientes que han completado su corte de cabello.
- Conteo de clientes actuales en la sala de espera.

Parámetros modificables durante ejecución:

- Frecuencia de llegadas de clientes.
- Tiempo que el barbero se toma para cortar el cabello de un cliente.

Problema 3: Video Juego Space Invaders

Se deberá desarrollar un video juego similar al clásico “Space Invaders” de 1978, pero en esta ocasión será para 2 jugadores, se aconseja ver videos, referencias o directamente jugar un poco con el videojuego en internet para tener una mejor idea de lo que se solicita.

Básicamente se tendrá una pantalla en la cual en la parte superior irán apareciendo naves enemigas que irán descendiendo por la pantalla, cada una de ellas con “2 puntos de vida”, estas deberán ir apareciendo de forma aleatoria en la parte superior de la pantalla y cada 25 segundos irán apareciendo con más frecuencia. En la parte inferior se tendrán 2 naves, las cuales son aliadas y controladas por los jugadores, estas naves solamente se podrán mover de forma horizontal. Los comandos serán las siguientes teclas, para la primera nave: “A” - Izquierda, “S” - Disparar, “D” - Derecha; para la segunda nave: “J” - Izquierda, “K” - Disparar, “L” - Derecha.

Al momento de disparar se lanza un rayo de la nave y al impactar con un enemigo, se descuenta 1 punto de vida de este, al impactar por segunda vez, la nave enemiga desaparece. Las naves aliadas tendrán 3 puntos de vida cada una, se pierde una vida si un enemigo impacta en una nave o si un enemigo no es destruido y escapa en la parte inferior de la pantalla. Si un jugador pierde todas sus vidas desaparece de la pantalla, pero su compañero puede continuar.

Las naves de los jugadores pueden moverse a cualquier distancia de forma horizontal, pero no pueden traslaparse, ni pasar una encima de la otra al moverse de forma horizontal, es decir no pueden colisionar y se restringen entre ellas el área en la cual pueden moverse.

Entregables:

- Código fuente de cada uno de los tres problemas, en 1 o varias aplicaciones.
- Archivos ejecutables .jar
- Manuales de usuario
- Manuales técnicos

Consideraciones

- En la calificación se requerirá al estudiante que explique su código, lógica de programación y el uso de las herramientas que utilizó; pero no se permitirá la modificación del mismo.
- El desarrollo se puede hacer de forma local y utilizando el editor de código de preferencia, si se permite el uso de Drag and Drop para crear la interfaz gráfica.
- Se tendrá en cuenta la representación correcta a través de la interfaz gráfica, no es necesario realizar diseños complejos, pero que si modelen todas las partes de interés del enunciado.
- Cualquier copia total o parcial será reportada a la Escuela de Sistemas para que proceda como corresponde.
- Los grupos de 3 integrantes deberán desarrollar el software para todos los problemas, los de 2 integrantes solamente el software de 2 problemas a elección, pero deberán realizar un análisis del tercero en la documentación con la información previamente citada, es decir indicando los casos en los que podría haber múltiples procesos, problemas que podrían darse y cómo se podrían resolver.

Penalizaciones

- Si el problema evaluado no puede pausarse, o no permite la modificación de los parámetros solicitados durante la ejecución, se tendrá una penalización del 30% sobre dicho problema.
- Entregas tarde tendrán una penalización del 35% sobre la nota obtenida.

Forma de Entrega

- Mediante UEDI subiendo el archivo zip: [SO2]Proyecto_<no_grupo>
- Subiendo todo a un repositorio de GitHub el cuál debe ser privado con el nombre: proyecto_grupo<no_grupo>_so2. Se deberá agregar al usuario: *brayan-chinchilla*.

Se calificarán a partir de los ejecutables enviados.

Fecha de Entrega:

Miércoles 27 de octubre de 2021 hasta las 23:59