Trabajo Práctico VIII: Concurrencia

Programación concurrente.
Aplicaciones
Creación de un Threads
Ciclo de vida un Thread
Extendiendo de la clase Thread
Implementando la interfaz Runnable
Compartiendo objetos entre distintos Threads
Sincronización de Threads
Métodos synchronized
Métodos estáticos sincronizados
Comunicación entre Threads: wait, notifyAll, notify
La sentencia synchronized.
Bloqueo selectivo
Semáforos
Métodos acquire y release

Objetivos de la práctica

Comprender la manera de no corromper la zona crítica de un recurso compartido en un ambiente concurrente.

Crear threads extendiendo de la clase Thread y crear threads extendiendo de la interfaz Runnable.

Sincronizar y comunicar Threads empleando la técnica de monitores.

Utilizar métodos sincronizados.

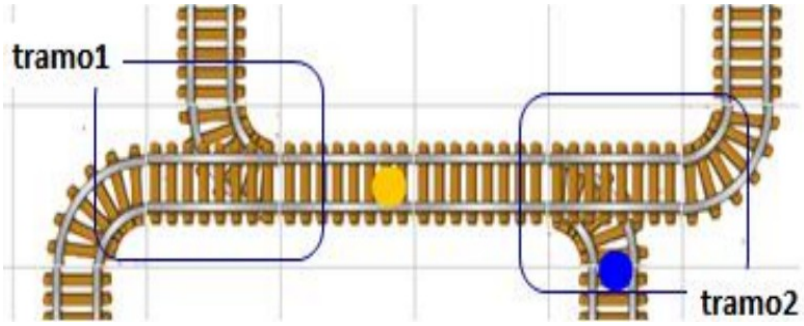
Utilizar la sentencia synchronized.

Aplicar técnicas de bloqueo selectivo.

Utilizar sermáforos.

Ejercicio 1

El tramo de vía compartido



Se tiene una zona de vía compartida por varios trenes y se desea controlar la entrada de trenes por uno y otro lado

Caso 1:

El monitor debe controlar que en el tramo de vía compartido nunca hay más de 1 tren en cada momento. Es decir, o está vacío o está ocupado con 1 tren.

Cuando un tren quiere entrar y hay otro tren dentro, queda esperando hasta que la vía esté libre.

Cuando un tren sale, se lo notifica a los trenes que estén esperando.

Caso 2:

Al igual que el caso 1 anterior, pero además, cuando un tren sale, tiene preferencia para entrar un tren que esté esperando en dirección opuesta.

Se trata de hacer que el recurso se comparta de forma equitativa.

Caso 3:

Se permite que haya varios trenes circulando en la misma dirección por el tramo compartido.

Se trata de optimizar el uso de un recurso compartido.

Caso 4:

Al igual que el caso 3, puede haber varios trenes circulando en el mismo sentido. Pero además, cuando un tren sale, tiene preferencia para entrar un tren que esté esperando en dirección opuesta. Es decir, que un nuevo tren sólo entra si no hay nadie esperando en sentido contrario y si no hay nadie circulando en sentido contrario.

Se trata de hacer que el recurso se comparta de forma equitativa.

Ejercicio 2

Organizador de turnos

Un grupo de personas tienen un número asociado y de acuerdo a ese número son atendidos en un mostrador, en forma cíclica (nadie se va, siguen siendo atendidos varias vueltas). El problema es que, cuando una persona ha sido atendida y comienza un nuevo turno, todos van hacia el mostrador y compiten por se atendidos. Será responsabilidad del empleado del mostrador, atender a quien le corresponde el turno (y poner a “dormir” al resto).

Ejercicio 3

Juego de cartas secuencial, pero usando Threads.

Varias personas, del mismo grupo de gente del ejercicio anterior, se ponen a jugar a un juego de cartas. Conservan el mismo comportamiento que antes, o sea, no quieren respetar su turno y quieren jugar en forma continua. De alguna forma, se debe ordenar el turno de cada uno y conservar la forma cíclica del juego.

El juego consiste en una versión simplificada de Black Jack. Gana la mano quien suma mayor cantidad de puntos sin sobrepasar los 21, si pasa los 21 la pierde. El juego finaliza cuando se terminan las cartas de la baraja y gana quien ha sumado más manos ganadas.

En su turno, cada jugador toma cartas de la baraja de a una hasta que decide “plantarse”. La baraja contiene 4 cartas que valen del 1 al 9 puntos, (cuatro cartas valen 1 puntos, cuatro valen 2 puntos, etc.) y dieciséis cartas que valen 10 puntos. Cada jugador desconoce los puntos que obtuvieron los demás jugadores hasta que la mano termina, momento en que se comparan los resultados de todo los jugadores y se decide quien gana (si existe empate, nadie gana al mano).

Ejercicio 4

Productor/Consumidor

Implementarlo utilizando synchronized, wait, notifyAll.

Implementar nuevamente utilizando Semaforos.

Sean las siguientes clases:

1. Libro (es parte del recurso compartido). Contiene los siguientes atributos:

   idLibro: número entero. (clave)

   Título: String.
2. Biblioteca (recurso compartido). Implementada en base a una tabla de hashing que contiene muchos Libros. Es responsable de la sincronización.
3. Socio (Thread o Runneable). Cada socio retira y luego devuelve el libro retirado. Debe simular un comportamiento usando números aleatorios (investigar) para elegir idLibro y el tiempo de espera para devolver y el tiempo de espera para volver a retirar.

Poner todo esto en funcionamiento con una biblioteca de 5 libros y 10 usuarios que compiten por el recurso compartido y esperan cuando no está el libro buscado en la biblioteca.

Ejercicio 5

Complicar un poco la cosa y agregar la siguiente complejidad:

a) Crear una clase Donador (Thread o Runnable) que agregar nuevos libros a la biblioteca. La dificultad está en que la biblioteca puede almacenar hasta 10 libros (puede tener más que 10, pero solamente 10 en sus estantes). O sea que cuando el donador quiera agregar un nuevo libro, deberá esperar hasta que alguien retire alguno, si es que no queda lugar en la biblioteca.

b) Aumentar el número de Socios para generar un escenario estable.

Ejercicio 6

Sincronización más granularizada.

a) La clase Biblioteca tiene dos tipos de objetos a prestar, libros y revistas. Crear dos clases de socios, SocioLibro y SocioRevista. Bloquear en forma selectiva según esta clasificación. O sea, que un socioLibro no interfiera con un socioRevista y viceversa.