INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

LABORATORIO: Sistemas Operativos

Práctica 9

Coordinación y cooperación entre procesos

LosDos

Integrantes:

Amanda Velasco Gallardo - 154415
Carlos Octavio Ordaz Bernal - 158525

Fecha(s) de elaboración de la práctica:

29 de marzo de 2019

Introducción

Los procesos concurrentes son procesos que coinciden o comparten, durante un periodo de tiempo específico, tiempo y espacio de CPU. Los procesos concurrentes pueden estar siendo ejecutados en un CPU o en varios, los cuales pueden tener uno o más núcleos, dentro de un sistema de cómputo.

Un programa concurrente es un conjunto de programas secuenciales ordinarios los cuales son ejecutados en paralelismo abstracto. El paralelismo es abstracto porque no requiere de un procesador por separado para cada proceso concurrente; no importa si uno o varios procesadores son los que realizan las operaciones de cada uno de los procesos.

La principal razón por la cual resulta importante o necesario trabajar con procesos concurrentes cooperantes es para compartir información entre procesos. Además, permite acelerar el cómputo de cálculos, y facilita modularizar la ejecución de un programa.

Un monitor es una estructura de alto nivel que provee sincronización segura entre los *threads* de una aplicación concurrente. La máquina virtual de `Java` permite que una aplicación pueda tener varios `threads` de ejecución corriendo concurrentemente. Un monitor tiene su propia clase y sus atributos sólo son accesibles desde los propios métodos de instancia del monitor. Un *thread* o proceso sólo puede tener acceso a un monitor invocando el método del mismo, uno a la vez; en consecuencia, sólo se puede estar ejecutando un monitor a la vez. En otras palabras, un monitor provee exclusión mutua entre los procesos concurrentes que lo invocan.

Todos los mecanismos de sincronización son proporcionados desde el interior de los monitores, dichos mecanismos son:

- Acceso a métodos del monitor cumpliendo con la exclusión mutua.
- Auto-suspensión de la ejecución de un thread, dentro de un método de un monitor, quedando dicho monitor disponible para otro *thread*.
- Reanudación de la ejecución de un *thread* auto-suspendido.

En `Java`, el modificador `synchronized` permite ejecutar código sincronizado para bloquear un objeto, sin necesidad de invocar el método sobre el otro objeto. Si se invoca este método desde un *thread*, el objeto queda bloqueado para otros *threads*. Si otro invoca otro método `synchronized`, sobre el mismo objeto, éste quedará suspendido hasta que se libere el bloqueo.

Desarrollo

1. Ejecutar el programa `ProCon` en diferentes corridas para 1 y 30 lugares de almacenamiento. Notar que siempre que algún thread consumidor reporta un bien, es porque la impresión del *thread* productor ya apareció.
En todas las distintas corridas se pudo observar dicho comportamiento.

2. Los métodos sincronizados `get` y `put` cada vez que se invocan, ¿cuántos bienes toman o ponen?
Solamente uno.

3. Ejecutar el programa `ProCon` en diferentes corridas para 1, 3, 20 y 30 lugares de almacenamiento. ¿En cuales de estas corridas seguro el almacén se llenará durante la corrida, en cuáles corridas puede que el almacén llegue a llenarse durante la corrida y en cuáles corridas jamás se llenará el almacén durante la corrida?
El almacén siempre se llenará con 1 lugar de almacenamiento. Puede que llegue a llenarse con 3 y 20 lugares de almacenamiento, pero con 30 lugares nunca se llenará ya que se están produciendo solamente 20 artículos.

4. Modificar en el programa del consumidor la instrucción `libHilos.hacerTiem(1,1000)` para que cada consumidor tarde más tiempo entre que toma y toma. Ejecutar `ProCon` para 20 lugares de almacenamiento, ¿qué diferencias se notan?
La ejecución se vuelve muchísimo más larga debido a los tiempos de espera de los consumidores, por lo que el almacén se llena o se acerca al tope rápidamente con los 20 productos y debe esperar mucho tiempo para vaciarse.

5. ¿Cuál o cuáles estructuras, de la aplicación *Productores - Consumidores*, corresponden a una estructura monitor?, ¿qué clase se implementa en el almacén?
La clase `Madriguera` es un monitor. La clase que se implementa dentro del almacén es `Pila`.

6. ¿Cuál es la utilidad de los métodos de instancia `vacío` y `lleno` en la clase `Pila`? ¿Para qué se usan?
`vacío` sirve para calcular si el tamaño actual de la pila es menor o igual a

cero, mientras que `lleno` sirve para calcular si el tamaño actual de la pila es mayor o igual al máximo soportado. Se usan dentro de la clase `Madriguera`. En particular, `vacío` se usa dentro del método `get()` de la Madriguera para que el consumidor espere por un artículo mientras el almacén esté vacío. `lleno` se usa dentro del método `put()` de la Madriguera para que el productor espere para colocar un artículo mientras el almacén se encuentre lleno.

7. ¿Bajo cuáles condiciones este programa concurrente podría quedarse bloqueado?
Los métodos con el modificador *synchronized* que exigen una cierta ejecución secuencial son los que podrían resultar en un bloqueo de los agentes que intentan acceder a los elementos de la Madriguera debido a los `wait()` dentro de dichos métodos. Por lo tanto, el programa podría quedarse bloqueado si, por ejemplo, un consumidor continúa esperando por un artículo en un almacén vacío en el cual los productores ya no colocarán más artículos. También ocurriría un bloqueo en el caso que un productor espera a poner un artículo en un almacén lleno cuando los consumidores no retirarán más artículos.

Conclusiones

El desarrollo de esta práctica nos permitió conocer de manera más detallada la forma en que los *hilos* se sincronizan por medio del método *synchronized* dentro la máquina virtual de `Java`. El modificar el código del ejercicio *Productores - Consumidores* nos dejó profundizar en la manera en que se crean los *hilos* y se sincronizan por el modificador de `Java`. Pudimos comparar que, a diferencia de ejercicios similares donde no se utilizan monitores y los *hilos* se ejecutan de manera secuencial o no realizan la tarea deseada, los *hilos* facilitan la coordinación, por ello es de gran importancia conocer la forma en que se sincronizan por medio del modificador *synchronized*.

Referencias

- Ríos, J. (2019). Notas del curso de Sistemas Operativos. Recuperado el 03 de abril de 2019, del sitio web: Comunidad ITAM.