Prácticas Concurrencia y Distribución (22/23)

Arno Formella, Alba Nogueira Rodríguez, David Ruano Ordás

semana 24 abril – 28 abril

Práctica 9: Productor/Consumidor

De la Wikipedia: (https://es.wikipedia.org/wiki/Problema_productor-consumidor)

> En computación, el problema del productor-consumidor es un ejemplo clásico de problema de sincronización de multiprocesos. El programa describe dos procesos, productor y consumidor, ambos comparten un buffer de tamaño finito. La tarea del productor es generar un producto, almacenarlo y comenzar nuevamente; mientras que el consumidor toma (simultáneamente) productos uno a uno. El problema consiste en que el productor no añada más productos que la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío.

Como escenario de tal problema nos podemos imaginar el servicio de los platos en un gran banquete: los pinches en la cocina preparan los platos y los colocan en una barra de cierto tamaño en la salida de la cocina, las camareras cogen los platos de la barra y los llevan a las mesas de los invitados. Obviamente no se puede colocar un plato si la barra está llena, ni coger uno si la barra está vacia.

La idea para una solución es la siguiente, ambos tipos de procesos (productores y consumidores) se ejecutan simultáneamente y se "despiertan" o "duermen" según el estado del búfer. Concretamente, un productor agrega productos mientras quede espacio y en el momento en que se llene el buffer se pone a "dormir". Cuando un consumidor toma un producto notifica que se puede comenzar a colocar otra vez. En caso contrario, si el búfer se vacía, un consumidor se pone a dormir y en el momento en que un productor agrega un producto crea una señal para despertar.

Esta solución se puede implementar usando diferentes mecanismos de comunicación entre procesos, aquí trabajamos con tres posibilidades.

1. **Productor/consumidor con `wait/notify/notifyAll`.**
   Objetivo: Implementar un búfer limitado (*bounded buffer*) y estudiar el problema productor/consumidor.

a) El código debe constar de tres clases: `Producer`, `Consumer` y una clase `Buffer`. La clase `Buffer` se usa para sincronizar las operaciones, "generar" y "coger" (implementadas como funciones miembro de la clase `Buffer`) que los hilos `Producer` y `Consumer` usan concurrentemente. La funcionalidad del código debe ser la siguiente:

- Desde el programa principal, se lanzan los hilos productores y consumidores con un objeto compartido de tipo `Buffer`, que contiene una lista enlazada con tipos enteros (que representan los platos).
- Un productor debe agregar valores al búfer (utilizando el método `write()`) y el consumidor debe eliminar valores (utilizando el método `read()`).
- El búfer tendrá una capacidad máxima que no se puede exceder (es decir, no puede contener más elementos que lo que indica dicha capacidad).
- Si un productor intenta agregar un valor cuando el búfer ha alcanzado su capacidad, debe esperar hasta que un consumidor (sincronizado con la condición `notFull`) deje espacio.
- Si un consumidor intenta coger un elemento cuando el búfer ha alcanzado su capacidad mínima, es decir, está vacio, debe esperar hasta que un productor (sincronizado con la condición `notEmpty`) genere un item.
- Tanto la producción de un item (pinche genera el plato) como la consumición de un item (camarera lleva el plato a la mesa) se simula con un breve tiempo de espera aleatoria. Para realizar tal espera mira las clases del manual de Java en `java.util.concurrent.ThreadLocalRandom` y/o `java.util.Random`.
- El número tanto de productores, como de consumidores, como de items colocables en el búfer se debe poder configurar via línea de comando.

b) Observa que en este problema se puede producir una situación llamado *bloqueo* (o *deadlock*), es decir, que dos (o más) procesos esperan a una situación que no se da nunca. Explica cómo puede suceder esto. Identifica la(s) línea(s) en tu código que podrían producir tal punto muerto potencial (en caso que existan).

2. **Problema del consumidor/productor con Java *locks***:

a) Modifica la clase de `Buffer` del problema de arriba para utilizar objetos basados en `ReentrantLock()` de Java y empleando la interfaz de `Condition`. Usa `lock/unlock` del objeto `Lock` y `await/signal` de la interfaz de `Condition` para lograr los mismos objetivos que conseguiste con el uso de `notify/wait` dentro de los bloques `synchronized`. (Mira la documentación de Java sobre la interfaz `Condition`).

3. **Problema del consumidor/productor con Java *collection objects***:

   a) Reemplace la clase `Buffer` con una cola con bloqueo (*blocking queue*) de la colección Java.

      - Una cola con bloqueo hace que un hilo se bloquee (es decir, pasar al estado de espera) cuando intenta agregar un elemento a una cola completa, o eliminar un elemento de una cola vacía. Permanecerá allí hasta que la cola ya no esté llena o no esté vacía. Hay tres implementaciones de colas de bloqueo en Java: `ArrayBlockingQueue`, `LinkedBlockingQueue`, and `PriorityBlockingQueue`.

   b) Modifica tu código para utilizar estos tipos *buffer*.