Prácticas Concurrencia y Distribución (17/18)

Arno Formella, Anália García Lourenço, Lorena Otero Cerdeira, David Olivieri

semana 2 – 8 abril

7. Semana 7 (02/04–08/04): Productor/Consumidor

From Wikipedia:

> 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.
>
> La idea para la solución es la siguiente, ambos procesos (productor y consumidor) se ejecutan simultáneamente y se "despiertan" o "duermen" según el estado del buffer. Concretamente, el productor agrega productos mientras quede espacio y en el momento en que se llene el buffer se pone a "dormir". Cuando el consumidor toma un producto notifica al productor que puede comenzar a trabajar nuevamente. En caso contrario si el buffer se vacía, el consumidor se pone a dormir y en el momento en que el productor agrega un producto crea una señal para despertarlo. Se puede encontrar una solución usando mecanismos de comunicación interprocesos, generalmente se usan semáforos.

1. **(P4: para entregar en grupo de práctica)**: Productor/ consumidor con `wait/notify`.
   Objetivo: Implementar un *bounded buffer* y estudiar el problema del Productor/Consumidor.

   a) **Configuración del problema:** El código consta de tres clases: `Productor`, `Consumidor` y una clase `Buffer`. La clase `Buffer` se usa para sincronizar dos operaciones, escribir y leer (implementadas como funciones miembro de la clase `Buffer`) que son utilizadas por los hilos `Productor` y `Consumidor`. La funcionalidad del código debe ser la siguiente:

      1) Desde el programa principal, lanzar los hilos Productor/Consumidor con un objeto compartido de tipo `Buffer` (llámelo buffer), que contiene una lista enlazada con tipos enteros.
      2) El productor debe aumentar el valor del búfer (utilizando el método `write()`) y el consumidor debe disminuir el valor (utilizando el método `read()`).
      3) La variable compartida tiene una capacidad máxima que el valor no puede exceder (valor entero `CAPACIDAD`)
      4) Si el productor intenta agregar un valor cuando el búfer ha alcanzado su capacidad, debe esperar al Consumidor (sincronizado con la condición `notFull`)
      5) La variable compartida tiene una capacidad mínima que el valor no puede aprobar (por ejemplo, puede tener el valor cero). Si el consumidor intenta disminuir el valor cuando el búfer ha alcanzado su capacidad mínima, debe esperar al productor (sincronizado con la condición `notEmpty`)

b) Este problema puede producir una situación llamada deadlock. Explica cómo puede suceder esto. Identifica la(s) línea(s) en tu código que podrían producir el punto muerto potencial.

2. **(P3: para entregar dentro de una semana)** Problema del consumidor/productor con Java *locks*:
Parte 1

   a) Modifica la clase de `Buffer` del problema de arriba para utilizar objetos de `ReentrantLock()` de Java y la interfaz de `Condition`. Debe usar `lock/unlock` del objeto `Lock` y `await/signal` de la interfaz de `Condition` para lograr los mismos objetivos que la `notify/wait` con el método `synchronized`. (Ver la documentación de Java sobre el interfaz `Condition`).

   b) Modifica el código para que un productor nunca escriba dos líneas consecutivas

3. **(P3: para entregar dentro de una semana)** Problema del consumidor/productor con Java *collection objects*:
Parte 2

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

      - Una cola de 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 más tiempo. Hay tres colas de bloqueo en Java: `ArrayBlockingQueue`, `LinkedBlockingQueue`, and `PriorityBlockingQueue`.

   b) Modifica tu código para utilizar este *buffer*.