Centro de Enseñanza Técnica Industrial

Desarrollo de Software

Principio básico del productor y consumidor

Jesús Alberto Aréchiga Carrillo

22310439 5N

Profesor

José Francisco Pérez Reyes

Octubre 2024

Guadalajara, Jalisco

Introducción

En esta práctica se analiza el comportamiento del código de principio básico del productor y consumidor múltiples veces. En el código existe un error aparente, se va a identificar y solucionar dicho error.

Desarrollo

Los códigos para implementar son los siguientes:

Productor:

```
public class Productor extends Thread{

    private final Contenedor contenedor;
    public Productor(Contenedor c){
        contenedor = c;
    }
    @Override
    public void run(){
        for(int i = 0; i < 10; i++){
            contenedor.put(i);
            System.out.println("Productor. put: " + i);
            try{
            sleep((int)(Math.random() * 100));
            } catch(InterruptedException e) { }
        }
    }

}
```

Consumidor:

```java
public class Consumidor extends Thread{

    private final Contenedor contenedor;
    public Consumidor(Contenedor c){
        contenedor = c;
    }
    @Override
    public void run(){
        int value;
        for(int i = 0; i < 10; i++){
            value = contenedor.get();
            System.out.println("Consumidor. get: " + value);
        }
    }

}
```

Contenedor:

```java
public class Contenedor {

    private int dato;
    private boolean hayDato = false;

    public synchronized int get(){
        while(hayDato == false){
            try{
                // Espera a que el productor coloque un valor
                wait();
            } catch(InterruptedException e){ }
        }
        hayDato = false;

        // Notificar que el valor ha sido consumido

        notifyAll();
        return dato;
    }
    public synchronized void put(int valor){
        while(hayDato == true){
            try{
                // Espera a que se consuma el dato
                wait();
            } catch(InterruptedException e) { }
        }
```

```
        dato = valor;
        hayDato = true;
        // Notificar que ya hay dato.
        notifyAll();
    }

}
```

ProductorConsumidorTest:

```
public class ProductorConsumidorTest{

    public static void main(String[] args){
        Contenedor c = new Contenedor();
        Productor produce = new Productor(c);
        Consumidor consume = new Consumidor(c);
        produce.start();
        consume.start();
    }

}
```

Ejecuciones del código:

```
Consumidor. get: 0
Productor. put: 0
Productor. put: 1
Consumidor. get: 1
Productor. put: 2
Consumidor. get: 2
Productor. put: 3
Consumidor. get: 3
Productor. put: 4
Consumidor. get: 4
Productor. put: 5
Consumidor. get: 5
Consumidor. get: 6
Productor. put: 6
Productor. put: 7
Consumidor. get: 7
Consumidor. get: 8
Productor. put: 8
Productor. put: 9
Consumidor. get: 9
```

```
Productor. put: 0
Consumidor. get: 0
Consumidor. get: 1
Productor. put: 1
Productor. put: 2
Consumidor. get: 2
Consumidor. get: 3
Productor. put: 3
Consumidor. get: 4
Productor. put: 4
Productor. put: 5
Consumidor. get: 5
Productor. put: 6
Consumidor. get: 6
Consumidor. get: 7
Productor. put: 7
Productor. put: 8
Consumidor. get: 8
Productor. put: 9
Consumidor. get: 9
```

```
Productor. put: 0
Consumidor. get: 0
Consumidor. get: 1
Productor. put: 1
Productor. put: 2
Consumidor. get: 2
Productor. put: 3
Consumidor. get: 3
Productor. put: 4
Consumidor. get: 4
Consumidor. get: 5
Productor. put: 5
Productor. put: 6
Consumidor. get: 6
Consumidor. get: 7
Productor. put: 7
Consumidor. get: 8
Productor. put: 8
Productor. put: 9
Consumidor. get: 9
```

```
Productor. put: 0
Consumidor. get: 0
Productor. put: 1
Consumidor. get: 1
Productor. put: 2
Consumidor. get: 2
Productor. put: 3
Consumidor. get: 3
Productor. put: 4
Consumidor. get: 4
Productor. put: 5
Consumidor. get: 5
Consumidor. get: 6
Productor. put: 6
Productor. put: 7
Consumidor. get: 7
Productor. put: 8
Consumidor. get: 8
Productor. put: 9
Consumidor. get: 9
```

Error o problema identificado:

Al utilizar notifyAll() se están despertando todos lo hilos que están esperando, incluyendo potencialmente el hilo que está ejecutando el método. Si esto sucede, lleva a que el consumidor pueda consumir datos que aún no se han producido, como se puede observar en todas las instancias que están capturadas arriba. Se puede ver cómo es que el consumidor consume cuando teóricamente el productor no ha producido.

Solución al problema:

Realmente no existe problema en la lógica del programa, si se está haciendo todo como debería, no existen condiciones de competencia y los productores y consumidores si hacen lo que tienen que hacer en el momento que lo tienen que hacer.

El verdadero error reside en la impresión en consola, ya que está fuera del método de get() y put(), el planificador hace su trabajo y le da los tiempos de ejecución a los hilos independientemente, haciendo que al salir del método, no sabremos cual va a ser la primera impresión en consola.

Por otro lado, si dejamos la impresión en consola dentro de los métodos get() y put() antes de notificar, primero va a hacer la impresión y luego va a notificar a los demás hilos para que sigan trabajando.

Códigos corregidos:

Productor:

```
public class Productor extends Thread{
    private Contenedor contenedor;
    public Productor(Contenedor c){
        contenedor = c;
        }
        public void run(){
        for(int i = 0; i < 10; i++){
            contenedor.put(i);
            try{
                sleep((int)(Math.random() * 100));
            } catch(InterruptedException e) { }
        }
    }
}
```

Consumidor:

```
public class Consumidor extends Thread{
    private Contenedor contenedor;
    public Consumidor(Contenedor c){
        contenedor = c;
    }
    public void run(){
        int value = 0;
        for(int i = 0; i < 10; i++){
            value = contenedor.get();
        }
    }
}
```

Contenedor:

```
public class Contenedor {
    private int dato;
    private boolean hayDato = false;

    public synchronized int get(){
        while(hayDato == false){
            try{
                // Espera a que el productor coloque un valor
                wait();
            } catch(InterruptedException e){ }
        }
        hayDato = false;

        // Notificar que el valor ha sido consumido
        System.out.println("Consumidor. get: " + dato);
        notifyAll();
        return dato;
    }
    public synchronized void put(int valor){
        while(hayDato == true){
            try{
                // Espera a que se consuma el dato
                wait();
            } catch(InterruptedException e) { }
        }
        dato = valor;
        hayDato = true;
        System.out.println("Productor. put: " + dato);
        // Notificar que ya hay dato.
        notifyAll();
    }
}
```

Como se puede ver, la impresión en consola está antes de notificar, ahora va a imprimir en consola y luego el programa sigue como debería.

```
Productor. put: 0
Consumidor. get: 0
Productor. put: 1
Consumidor. get: 1
Productor. put: 2
Consumidor. get: 2
Productor. put: 3
Consumidor. get: 3
Productor. put: 4
Consumidor. get: 4
Productor. put: 5
Consumidor. get: 5
Productor. put: 6
Consumidor. get: 6
Productor. put: 7
Consumidor. get: 7
Productor. put: 8
Consumidor. get: 8
Productor. put: 9
Consumidor. get: 9
```

Ahora el consumidor imprime en consola sólo si el productor ya lo hizo.