Análisis de Solución para Ejercicio de Programación Concurrente

Análisis Automático

November 7, 2025

1 Análisis del Código Propuesto

El programa proporcionado es un intento completo y complejo para resolver un enunciado desafiante de programación concurrente. La estructura general del problema está bien capturada; sin embargo, el código presenta varios errores críticos de lógica y de implementación de la concurrencia que impedirán su correcto funcionamiento y, en varios puntos, causarán un **deadlock**.

A continuación, se presenta un análisis detallado, fase por fase, comparándolo con los requisitos del enunciado.

1.1 Veredicto Rápido

- **Estructura General:** Buena. Se han identificado correctamente los roles (Recolector, Coordinador) y las fases del problema.
- **Lógica de Comunicación:** Incorrecta y propensa a deadlocks. El protocolo de envío y recepción de mensajes no está bien sincronizado.
- **Lógica de Concurrencia (Bloqueo):** La implementación del depósito es extremadamente ineficiente y podría ser problemática.
- **Lógica de los Procesos:** Algunos procesos clave, como `escalera`, no implementan lo que pide el enunciado.

2 Análisis Detallado de Errores Críticos

2.1 Fase 1: Recolección y Depósito

2.1.1 El Coordinador envía esquinas de depósito DIFERENTES (Error Grave)

El enunciado implica una única esquina de depósito compartida ("una esquina aleatoria"). Sin embargo, el código del coordinador genera y envía dos esquinas aleatorias diferentes, una para cada robot. Esto rompe el requisito de un recurso compartido y elimina la necesidad de sincronización con `BloquearEsquina`.

Código del Coordinador (Erróneo):

```
repetir 2
  Random(num,40,45)
  EnviarMensaje(num,r1) // Env a una esquina a r1
  Random(num,40,45)
  EnviarMensaje(num,r2) // Env a OTRA esquina a r2
```

Solución: Generar **una** esquina aleatoria para la avenida y **una** para la calle, y enviar esas mismas coordenadas a **ambos** robots.

2.1.2 El Proceso `dejarAlternado` es Extremadamente Ineficiente

Este proceso implementa una lógica de depósito muy costosa. Por cada objeto en la bolsa, el robot realiza un ciclo completo de bloqueo, viaje, depósito, viaje de vuelta y liberación. Si un robot recoge 50 objetos, hará 50 viajes de ida y vuelta, destruyendo la concurrencia.

Solución: El proceso de depósito debería seguir este flujo:

1. Bloquear la esquina **una vez**.
2. Ir a la esquina.
3. Depositar **todos** los objetos.
4. Volver a su posición.
5. Liberar la esquina **una vez**.

2.2 Fase 2: Limpieza y Análisis del Coordinador

2.2.1 Lógica de Recepción de Resultados Incorrecta (Deadlock)

El código para recibir los resultados en el coordinador es incorrecto y causará un deadlock. **Código del Coordinador (Erróneo):**

```
repetir 2
  RecibirMensaje(num,*) // Espera un mensaje de CUALQUIERA
  si (num=2)
    RecibirMensaje(num,r2) // ERROR! Si num=2, el mensaje vino de r1
  si (num=3)
    RecibirMensaje(num,r1) // ERROR! Si num=3, el mensaje vino de r2
```

Problema: Hay un cruce en la lógica. Si el coordinador recibe el ID `2` (de `r1`), intenta recibir el siguiente mensaje de `r2`, que aún no ha enviado nada. Mientras tanto, `r2` envía su ID (`3`), pero el coordinador no lo está escuchando. Esto resulta en un deadlock.

Solución: Implementar el protocolo de recepción robusto, donde primero se identifica al emisor y luego se escucha específicamente a ese emisor.

```
// Soluci n para recibir los totales
repetir 2
  RecibirMensaje(idRecibida, *)
  si idRecibida = 2 // El ID de r1
    RecibirMensaje(elem1, r1)
  sino // si idRecibida = 3 (el ID de r2)
    RecibirMensaje(elem2, r2)
```

Además, la lógica de recepción de resultados está duplicada en el código original, lo que agrava el problema.

2.3 Fase 3: La Carrera en Escalones

2.3.1 El Proceso `escalera` es Incorrecto

El enunciado pide un "recorrido en escalones de 1 en 1", que es un patrón de movimiento y giros. El proceso `escalera` implementado realiza giros, movimientos rectos y teletransportaciones, lo cual no cumple el requisito. **Solución:** El proceso debe implementar la lógica de un escalón, por ejemplo: `repetir valor { mover; derecha; mover; derecha; derecha; derecha; }`.

2.3.2 Cálculo de Escalones Incorrecto

El código calcula la cantidad de escalones basándose en los objetos que el **coordinador** limpió. El enunciado especifica que debe ser "la mitad de los objetos que limpió el **robot que juntó menos objetos**". **Solución:** Se debe utilizar la variable que contiene el total del perdedor (`elem1` o `elem2`, el que sea menor) para realizar el cálculo.

2.3.3 Detección del Ganador de la Carrera Incorrecta

El coordinador recibe dos mensajes, pero los recolectores envían un valor constante (`1`) que no los identifica. Por lo tanto, el coordinador no puede saber quién envió el mensaje primero. **Solución:** Cada recolector debe enviar su ID único (`2` o `3`). El coordinador espera el primer mensaje con `RecibirMensaje(idGanador, *)` para determinar al vencedor.

3 Resumen de Cambios Necesarios

Para que el programa cumpla la consigna, se requiere una reestructuración importante, especialmente en el robot `Coordinador`:

1. **Coordinador - Fase 1:** Generar **una** única esquina aleatoria y enviarla a **ambos** recolectores.
2. **Recolector - Fase 1:** Reescribir el proceso de depósito para que sea eficiente (bloquea una vez, deposita todo, libera una vez).
3. **Coordinador - Fase 2:** Arreglar la lógica de recepción de mensajes para evitar el deadlock, usando un protocolo de identificación explícito.
4. **Coordinador - Fase 3:**
   - Identificar correctamente el total de objetos del robot perdedor.
   - Usar ese total para calcular la cantidad de escalones.
   - Enviar los parámetros correctos (posición de inicio y cantidad de escalones) a cada recolector.
5. **Recolector - Fase 3:**
   - Reescribir el proceso `escalera` para que funcione correctamente.
   - Enviar su ID único al coordinador al finalizar.
6. **Coordinador - Fase 3 Final:** Esperar un solo mensaje con ID para determinar al ganador de la carrera.

El programa es un buen borrador, pero la complejidad del enunciado requiere que el protocolo de comunicación y la lógica de concurrencia sean impecables para evitar bloqueos.