Práctica 2. Puente de Ambite.

Presento los algoritmos implementados en el pseudocódigo del libro de Ben-Ari "Principles of concurrent and distributed programs".

Algoritmo 1.

```
monitor Puente
    integer peatones ← 0
    integer vehiculosNorte ← 0
    integer vehiculosSur ← 0
    condition SoloP, SoloVN, SoloVS

operation esperarCruzarP
    if vehiculosNorte ≠ 0 and vehiculosSur ≠ 0
        waitC(SoloP)
    peatones ← peatones + 1
    signal(SoloP)

operation esperarCruzarVN
    if peatones ≠ 0 and vehiculosSur ≠ 0
        waitC(SoloVN)
    peatones ← peatones + 1
    signal(SoloVN)

operation esperarCruzarVS
    if peatones ≠ 0 and vehiculosNorte ≠ 0
        waitC(SoloVS)
    peatones ← peatones + 1
    signal(SoloVS)
```

```
operation salirPuenteP
    peaton ← peaton - 1
    if peaton = 0
        Signal C (SoloVN)

operation salirPuenteVN
    vehiculoNorte ← vehiculoNorte - 1
    if vehiculoNorte = 0
        Signal C (SoloVS)

operation salirPuenteVS
    vehiculoSur ← vehiculoSur - 1
    if vehiculoSur = 0
        Signal C (SoloP)
```

---

peatón	vehiculoNorte	vehiculoSur
p1: Puente.esperarCruzarP	p1: Puente.esperarCruzarP	p1: Puente.esperarCruzarP
p2: Cruzar puente	p2: Cruzar puente	p2: Cruzar puente
p3: Puente.salirPuenteP	p3: Puente.salirPuenteP	p3: Puente.salirPuenteP

Se puede observar que en el algoritmo anterior los métodos y las variables condición del monitor están asociadas a una de las tres clases de usuario de forma que cada clase y sus respectivas variables y operaciones se comportan de forma simétrica.

El monitor se encarga de la seguridad del algoritmo, es decir, de que en todo momento sólo haya usuarios de una sola clase. En efecto, sean P, N y S variables correspondientes respectivamente a los peatones, vehículos procedentes del norte y vehículos procedentes del sur que se encuentran cruzando el puente en cada momento se tienen los siguientes invariantes.

$P \geq 0$, $N \geq 0$, $U \geq 0$, P = peatones, N = vehículos Norte

S = vehículos Sur

$$(P>0 \rightarrow (N=0 \wedge S=0)) \wedge (N>0 \rightarrow (P=0 \wedge S=0)) \wedge (S>0 \rightarrow (P=0 \wedge N=0))$$

Las seis primeras fórmulas son evidentes puesto que las variables del monitor aumentan y disminuyen conforme los usuarios de las clases asociadas entran y salen del puente y es claro que el número de usuarios en el puente no puede ser negativo.

Como inicialmente el puente se encuentra vacío las tres partes de la última fórmula son ciertas.

La primera parte de la conjunción $(P \rightarrow (N=0 \wedge S=0))$ sólo puede hacerse falsa aumentando P cuando o N o S sean positivas. La única instrucción que aumenta P es esperarCruzar, pero si no se cumpliera $N \neq 0 \wedge S \neq 0$ entonces el proceso quedaría bloqueado en la variable condición SoloP, por lo que sólo se podría efectuar el incremento de P una vez el proceso fuese liberado por la instrucción SignalC(SoloP) de salirVS, pero esta instrucción sólo se ejecuta si S=0 y puesto que SalirVS tiene precondición S>0, el propio invariante nos asegura que N=0 también se cumple. Puesto que ni SalirVS ni esperarCruzarP modifican N, se tiene que $N=0 \wedge S=0$ se cumple cuando se libera un proceso bloqueado en SoloP.

salirP sólo disminuye P en una unidad cuando es positivo lo que no puede hacer falsa ninguna de las partes del invariante.

Por simetría se tiene mediante un razonamiento análogo que las operaciones asociadas a las otras clases de usuarios tampoco hacen falso el invariante.

La solución anterior no tiene deadlocks pero sufre de inanición. Supongamos (como haremos en la simulación implementada) que los peatones tardan más tiempo en cruzar que lo que tarda un nuevo peatón en llegar al puente. Entonces antes de que un peatón terminará de cruzar el puente entrarían nuevos peatones recien llegados. El flujo constante de peatones provoca inanición de los vehículos procedentes de ambos lados que no podrían cruzar.

Para resolver el problema se modifica el algoritmo añadiendo un nueva variable condición a cada clase de usuario. Se tratará sólo el caso de los peatones ya que los clases se comportan de forma simétrica.

Se añade al monitor:

```
condition PeatonesNoCruzando

operation esperarCruzarP
    if Peatones ≠ 0
        waitC(PeatonesNoCruzando)
    if vehiculosNorte ≠ 0 and vehiculosSur ≠ 0
        waitC(SoloP)
    peatones ← peatones + 1
    signal(SoloP)
```

```
operation salirPuenteP
    peaton ← peaton - 1
    if peatón = 0
        signalC (SoloVN)
        signalC (PeatonesNoCruzando)
```

El efecto de esta modificación del monitor es que ahora se impide que comience a cruzar un peatón que llegue al puente mientras otros peatones estaban cruzando, forzándole a esperar al siguiente turno de peatones, de forma que se evita el flujo continuo de peatones y la inanición de vehículos.

El argumento de la seguridad del primer algoritmo no se ve modificado para el segundo.

Es evidente que no hay deadlocks ya que las distintas clases de usuarios se van turnando cíclicamente de forma que siempre hay una clase de usuarios que puede cruzar.