**Práctica 1 – Variables compartidas**

**1. Para el siguiente programa concurrente suponga que todas las variables están inicializadas en 0 antes de empezar.**

**Indique cual/es de las siguientes opciones son verdaderas:**

1. En algún caso el valor de x al terminar el programa es 56.
2. En algún caso el valor de x al terminar el programa es 22.
3. En algún caso el valor de x al terminar el programa es 23.
4. X puede obtener un valor incorrecto por interferencias.

|  |  |  |
| --- | --- | --- |
| **P1::** If (x = 0) then  y:= 4\*2;  x:= y + 2; | **P2::** If (x > 0) then  x:= x + 1; | **P3::** x:= (x\*3) + (x\*2) + 1; |

Primero ejecuta todo el 1, después todo el 2, y después el 3, da 56

Arranca el proceso 3, queda el valor 0 porque hace 0\*3, después hace p1, después vuelve a p3 y queda 21, y después va a p2 y queda 22.

Arranca el proceso 3, hace 0\*3, después va al proceso 1, y queda en X el valor 10, después va al proceso 2, y queda el valor 11, luego continua con el proceso 3 , 11\*2= 22 + 1= 3

**Todas las opciones son verdaderas.**

Interferencias se pueden dar porque por ejemplo podría arrancar el proceso 3, no finalizar y continuar el 1.

**P1**

**y= 4\*2**

(1.1) Load 4, Acumulador

(1.2) Mult 2, Acumulador

(1.3) Store Acumulador, X

**x:= y + 2**

(2.1) Load Y, Acumulador

(2.2) Add 2, Acumulador

(2.3) Store Acumulador, X

**P2**

**x:= x + 1;**

(3.1) Load X, Acumulador

(3.2) Add 1, Acumulador

(3.3) Store Acumulador, X

**P3**

**x:= (x\*3) + (x\*2) + 1;**

(4.1) Load X, Acumulador

(4.2) Mult 3, Acumulador

(4.3) Load X, Acumulador

(4.4) Mult 2, Acumulador

(4.5) Add 1, Acumulador

(4.6) Store Acumulador, X

**2.Dado un número N verifique cuántas veces aparece ese número en un arreglo de longitud M. Realice el algoritmo en forma concurrente utilizando <> y . Escriba las condiciones que considere necesarias.**

array int [1..M] arreglo;

int cantidad=0;

process contar(int N) {

for i=1 to M {

if(arreglo[i] == N) then {

**< cantidad ++; >**

}

}

}

**<e>** indica que la expresión **e** debe ser evaluada atómicamente.

La solución óptima es dividir el arreglo, por ejemplo en dos partes, y que cada proceso analice cada parte , concurrentemente. Cada proceso , debe tener una variable local interna, que cuenta y luego se suma el valor almacenado a la variable compartida.

int cantidad = 0;

array [1..M] arregloNumeros;

int N;

**process contar[i=0..3]** {

**var**

int cant= 0;

**begin**

posInicial = calcularIndice();

for j= posInicial to posFinal {

if(arregloNumeros[j] == N) {

cant++;

}

}

< cantidad = cantidad + cant; >

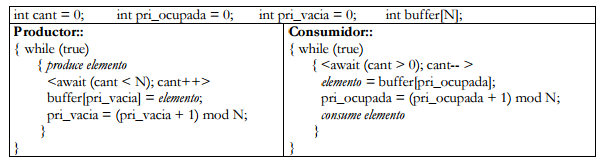
}

**3. En base a lo visto en la clase 3 de teoría.**

**a) Indicar si el siguiente código funciona para resolver el problema de Productor/Consumidor con un buffer de tamaño N. En caso de no funcionar, debe hacer las modificaciones necesarias.**

La exclusión mutua sería hasta después del MOD N, porque podría pasar que antes de guardar el elemento en el buffer, llegue el consumidor, y si no se guardó el elemento en el buffer y se aumentó *cant*, el consumidor no va a sacar nada del buffer.

Además, si no pongo la linea de *buffer…* y *pri\_vacia…*  adentro de la exclusión mutua, podría ser que venga otro proceso y me modifique los valores, lo que sería incorrecto, ya que se incrementó la cantidad (o decrementó en el caso del consumidor), por lo que se debe también guardar el elemento , o consumirlo, y actualizar *pri\_vacia* y *pri\_ocupada* sin que otro proceso pueda avanzar hasta que esto no termine.

****

**b) Modificar el código para que funcione para C consumidores y P productores.**