**PROGRAMACIÓN CONCURRENTE Y AVANZADA**

CUADERNO DE PRÁCTICAS

P05. Semáforos

**Semáforos en jBACI**

En jBACI, el intérprete tiene el tipo predeclarado SEMAPHORE. Un SEMAPHORE es una variable INTEGER no negativa que puede accederse solo a través de operaciones predefinidas. El semáforo binario, solo permite valores 0 y 1, está soportado por el subtipo BINARYSEM.

**Inicializando un semáforo**

La asignación de un valor a una variable de tipo SEMAPHORE o BINARYSEM únicamente se permite cuando se declara la variable. Por ejemplo:

VAR s: SEMAPHORE := 17;

b: BINAYSEM := 0;

Para inicializar un semáforo en otro momento que no sea la declaración, se puede utilizar la primitiva:

INITIALSEM (semaphore, integer\_expression);

En la llamada, el parámetro integer\_expression puede ser cualquier expresión que se evalúe a un valor entero válido para ese tipo de semáforo (no negativo para SEMAPHORE y 0 ó 1 para BINARYSEM). Por ejemplo, las siguientes dos llamadas a INITIALSEM son equivalentes a la inicialización de los semáforos del párrafo anterior:

INITIALSEM (s, 17);

INITIALSEM (b, 0);

**Procedimientos WAIT (P) y SIGNAL (V)**

Los procedimientos WAIT y SIGNAL los utilizan los procesos que se ejecutan concurrentemente con el objeto de sincronizarse. Ambas operaciones son atómicas. La definición es:

PROCEDURE WAIT (VAR s: SEMAPHORE);

(\* if s > 0, decrementa s en 1 y continua el llamador \*)

(\* if s = 0, bloquea al llamador \*)

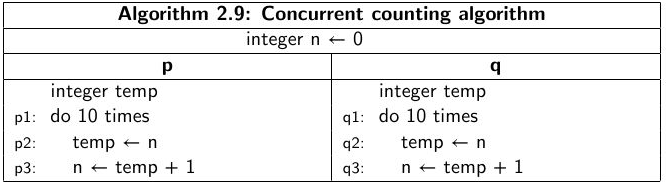
PROCEDURE SIGNAL (VAR s: SEMAPHORE);

(\* if s = 0 y uno o más procesos están esperando en s, despierta a uno de ellos (de forma aleatoria). Si no hay nadie esperando, incrementa s en 1. En cualquier caso, el llamador continua \*)

Para los semáforos binarios, una operación SIGNAL sobre un semáforo cuyo s es 1, no tiene efecto.

**Ejemplo de uso**

En la práctica 4, se escribió un programa para soportar el algoritmo 2.9:



La ejecución de dicho programa era no determinista por la existencia de una condición de carrera en el acceso a la variable compartida n. Se preguntó por un escenario tal que el resultado final de n no fuera 20, sino 10. A continuación se usa un semáforo binario para conseguir la exclusión mutua en el acceso a n y una ejecución determinista cuyo resultado siempre es 20.

program AddSem;

{ Add 10 to a variable in each of two processes. }

{ With a semaphore ensuring mutual exclusion, the answer is 20. }

var sum: integer := 0;

s : binarysem := 1;

procedure add10;

var i: integer;

local: integer;

begin

for i := 1 to 10 do

begin

wait(s);

local := sum;

sum := local + 1;

signal(s)

end

end;

begin

cobegin

add10;

add10;

coend;

writeln('Sum = ', sum);

end.

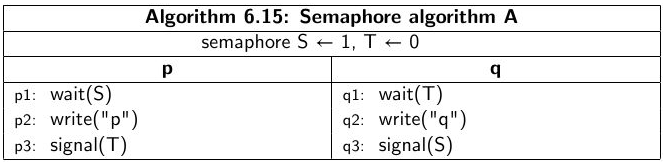
En el entorno jBACI hay descrito un programa para el problema de los lectores y escritores y otro para el problema de los filósofos. Prueba a ejecutarlos al igual que el programa anterior.

**Semáforos en Scala**

Scala utiliza la definición de semáforos de java (Semaphore class) contenida en el paquete java.util.concurrent. La semántica de estos semáforos es algo distinta a la que hemos visto en el libro de texto y por eso no incidiremos en este tipo de semáforos.

**Ejercicios**

**1.** **[AU]** Considera el siguiente algoritmo:



El programa equivalente en jBACi es el siguiente:

program Eje61;

VAR s: SEMAPHORE:= 1;

t: SEMAPHORE:= 0;

PROCEDURE p;

BEGIN

WAIT (s);

writeln ("p");

SIGNAL (t)

END;

PROCEDURE q;

BEGIN

WAIT (t);

writeln ("q");

SIGNAL (s)

END;

BEGIN

COBEGIN

p;

q;

COEND

END.

1. ¿Cuáles son las posibles salidas de este algoritmo?

**p q** ya que está restringido por el semáforo S y T.

1. ¿Cuáles son las posibles salidas si borramos la sentencia wait(s)?

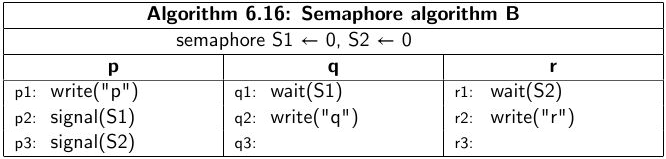
**p q** ya que es el único semáforo que \_funcionaría\_, solo entraría p, liberaría el semáforo t e imprimiría q

1. ¿Cuáles son las posibles salidas si borramos la sentencia wait(t)?

**p q** o **q p** ya que las dos pueden entrar primero

Razona las respuestas anteriores y compruébalas introduciendo el algoritmo en jBACI.

**2.** **[AU]** Considera el siguiente algoritmo:



El programa equivalente en jBACi es el siguiente:

program Eje62;

VAR s1, s2: SEMAPHORE:= 0;

PROCEDURE p;

BEGIN

writeln ("p");

SIGNAL (s1);

SIGNAL (s2)

END;

PROCEDURE q;

BEGIN

WAIT (s1);

writeln ("q");

END;

PROCEDURE r;

BEGIN

WAIT (s2);

writeln ("r");

END;

BEGIN

COBEGIN

p;

q;

r;

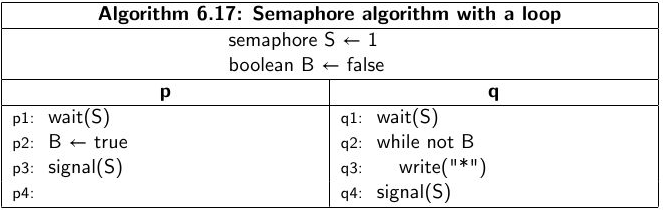
COEND

END.

¿Cuáles son las posibles salidas de este algoritmo? Razona la respuesta y compruébala introduciendo el algoritmo en jBACI.

**p q r / p r q**

**3. [AU]** Considera el siguiente algoritmo:



El programa equivalente en jBACi es el siguiente:

program Eje63;

VAR s: SEMAPHORE := 1;

b: BOOLEAN;

PROCEDURE p;

BEGIN

WAIT (s);

b:= TRUE;

SIGNAL (s);

END;

PROCEDURE q;

BEGIN

WAIT (s);

WHILE NOT b DO write ("\*");

SIGNAL (s);

END;

BEGIN

b:= false;

COBEGIN

p;

q;

COEND

END.

¿Cuáles son las posibles salidas de este algoritmo? Razona la respuesta y compruébala introduciendo el algoritmo en jBACI.

\* de por vida / nothing

**4.** **[LA]** Sean 4 procesos A, B, C y D. Los queremos ejecutar concurrentemente tal que satisfagan las siguientes condiciones:

* El proceso A debe ejecutarse antes que el proceso B
* El proceso C debe ejecutarse antes que el proceso D
* El proceso C debe ejecutarse antes que el proceso B

Programe en jBACI la sincronización anterior (utilizando semáforos) siendo que:

program Eje51;

VAR …

PROCEDURE A;

BEGIN

writeln ("Soy el proceso A");

END;

PROCEDURE B;

BEGIN

writeln ("Soy el proceso B");

END;

PROCEDURE C;

BEGIN

writeln ("Soy el proceso C");

END;

PROCEDURE D;

BEGIN

writeln ("Soy el proceso D");

END;

BEGIN

COBEGIN

…

COEND;

END.

**5. [LA]** Sea el siguiente programa en jBACI:

program Eje52;

PROCEDURE P1;

BEGIN

writeln ("Sentencia 1 de P1");

(\* 1 \*)

writeln ("Sentencia 2 de P1");

END;

PROCEDURE P2;

BEGIN

writeln ("Sentencia 1 de P2");

(\* 2 \*)

writeln ("Sentencia 2 de P2");

END;

BEGIN

COBEGIN

P1;

P2;

COEND

END.

Completar el programa para que se cumplan las siguientes condiciones de sincronización:

* Si P1 llega a (\*1\*) antes que P2 a (\*2\*), P1 esperará a que P2 llegue a (\*2\*)
* Si P2 llega a (\*2\*) antes que P1 a (\*1\*), P2 esperará a que P1 llegue a (\*1\*)

**6. [AV]** Se tiene un sistema con tres procesos fumadores y un proceso agente. Cada fumador está continuamente liando un cigarrillo y después se lo fuma. Para liar y fumar un cigarrillo, el fumador necesita tres ingredientes: tabaco, papel y cerillas. Uno de los procesos fumadores tiene tabaco, otro papel y el tercero tiene cerillas. El agente tiene una cantidad infinita de los tres materiales. El agente deja dos de los ingredientes en una mesa. El fumador que tiene el ingrediente que falta lía y se fuma un cigarrillo, avisándole al agente cuando termina. Entonces, el agente pone otros dos de los tres ingredientes en la mesa y el ciclo se repite. Escribir un programa, en jBACI, que sincronice al agente y a los fumadores utilizando semáforos.