# *El problema de los Filósofos*

## Solución con Semáforos

Para evitar una situación de interbloqueo se limita el número de filósofos en disposición de comer a 4.

```
PROGRAM Filósofos;
    VAR     tenedores: ARRAY [0..4] OF SHEMAPHORE;
            puerta: SEMAPHORE;
            i: INTEGER

    PROCEDURE Filósofo (i: INTEGER);
        BEGIN
            REPEAT
                … pensar …;
                WAIT (puerta);
                WAIT (tenedores [i]);
                WAIT (tenedores [i + 1] MOD 5);
                … comer …;
                SIGNAL (tenedores [i]);
                SIGNAL (tenedores [i + 1] MOD 5);
                SIGNAL (puerta)
            FOREVER
        END;

    BEGIN
        FOR i := 0  TO 4 INIT (tenedores [i], 1);
        INIT (puerta, 4);
        COBEGIN
            Filósofo(1); …; Filósofo(4)
        COEND
    END.
```

## Solución con Región Crítica Condicional

```
PROGRAM Filósofos;
    VAR     tenedores: SHARED ARRAY [0..4] OF 0..2;          (* Indica el nº de tenedores disponibles. *)
            i: INTEGER

    PROCEDURE Filósofo (i: INTEGER);
        BEGIN
            REPEAT
                … pensar …;
                REGION tenedores DO BEGIN
                    AWAIT (tenedores[i] = 2);                (* Espera a tener los dos tenedores *)
                    (tenedores [i + 1] MOD 5) := (tenedores [i + 1] MOD 5) – 1;          (* Los coge. *)
                    (tenedores [i + 4] MOD 5) := (tenedores [i + 4] MOD 5) – 1
                END;
                … comer …;
                REGION tenedores DO BEGIN
                    (tenedores [i + 1] MOD 5) := (tenedores [i + 1] MOD 5) + 1;          (* Los deja. *)
                    (tenedores [i + 4] MOD 5) := (tenedores [i + 4] MOD 5) + 1
                END
            FOREVER
        END;

    BEGIN
        FOR i := 0  TO 4 DO tenedores [i] := 2;
        COBEGIN
            Filósofo(0); …; Filósofo(4)
        COEND
    END.
```

## Solución con Monitores

```
PROGRAM Filósofos;
    TYPE filosofar = MONITOR;
        VAR     estado: (Pensando, Comiendo, Hambriento);
                espera: ARRAY [0..4] OF QUEUE

        PROCEDURE ENTRY CogePalillos (i: INTEGER);
            BEGIN
                estado[i] := Hambriento;
                prueba[i];
                IF estado[i] := Hambriento THEN
                    DELAY (Espera[i]);
            END;

        PROCEDURE ENTRY DejaPalillos (i: INTEGER);
            BEGIN
                estado[i] := Pensando;
                prueba [i + 1 mod 5];       (* Le da el palillo al filósofo de la derecha si lo necesita. *)
                prueba [i – 1 mod 5]        (* Le da el palillo al filósofo de la izquierda si lo necesita. *)
            END;

        PROCEDURE Prueba (i: INTEGER);
            BEGIN
                IF   estado(i – 1 mod 5) # Comiendo AND      (* Si el de la derecha no come y *)
                     estado(i + 1 mod 5) # Comiendo AND      (* el de la izquierda no come y *)
                     estado[i] = Hambriento THEN BEGIN       (* el del centro quiere comer. *)
                         estado [i] := Comiendo;
                         CONTINUE (espera[i])
                END
            END;

        VAR i: INTEGER;

        BEGIN
            FOR i := 0 TO 4 DO estado[i] := Pensando;
        END

    VAR fil: filosofar;

    PROCEDURE Filósofo (i: INTEGER);
        BEGIN
            REPEAT
                … pensar …
                fil . CogePalillos(i);
                … comer …
                fil . DejaPalillos(i)
            FOREVER
        END;

    BEGIN
        COBEGIN
            Filósofo(0); …; Filósofo(4)
        COEND;
    END.
```

# *El problema de los Coches en el Puente*

*En una carretera por la que circulan coches en los dos sentidos hay un puente donde sólo es posible la circulación en un sentido. Sobre el puente pueden haber varios coches circulando simultáneamente, pero en el mismo sentido.*

## Solución con Región Crítica

Esta solución contempla varios puntos:

1. El puente lo toma el primero que llega.
2. Se aplica un turno cuando hay coches en ambos sentidos.
3. Se tiene en cuenta si hay coches esperando en alguno de los lados del puente.

```
PROGRAM Río;
    VAR     puente:  SHARED RECORD
                         dentroN, dentroS : CARDINAL;          (* Coches, norte o sur, en el puente. *)
                         pasadosN, pasadosS: CARDINAL;         (* Coches que han pasado consecutivos. *)
                         turnoN, turnoS: BOOLEAN;              (* Indica de quién es el turno. *)
                         esperandoN, esperandoS: CARDINAL      (* Coches que esperan en cada sentido. *)
                     END;

    PROCEDURE CochesNorte (i: INTEGER);
        BEGIN
            REGION puente DO BEGIN
                esperandoN := esperandoN + 1;
                AWAIT ((dentroS = 0) & (turnoN OR esperandoS = 0));    (* No hay coches en S esperando. *)
                esperandoN := esperandoN – 1;
                dentroN := dentroN + 1;
                pasadosN := (pasadosN + 1) MOD 10;            (* Nº máximo de coches en un sentido (10). *)
                IF pasadosN = 0 THEN BEGIN                    (* Si es el último cambia el turno. *)
                    turnoN := FALSE;
                    turnoS := TRUE
                END
            END;
            … cruzar el puente …;
            REGION puente DO
                dentroN := dentroN – 1;
        END;

    PROCEDURE CochesSur (i: INTEGER);
        (* --- procedimiento simétrico a CochesNorte --- *)

    BEGIN
        WITH puente DO BEGIN
            dentroS := 0; dentroN := 0;
            pasadosS := 0; pasadosN := 0;
            turnoN := TRUE; turnoS := TRUE;
            esperandoN := 0; esperandoS := 0
        END;
        COBEGIN
            CochesNorte (1); …; CochesNorte (n);
            CochesSur (1); …; CochesSur (m)
        COEND
    END.
```

## Solución con Eventos

```
PROGRAM Río;
    VAR    puente:  SHARED RECORD
                        dentroN, dentroS : CARDINAL;          (* Coches, norte o sur, en el puente. *)
                        pasadosN, pasadosS: CARDINAL;         (* Coches que han pasado consecutivos. *)
                        turnoN, turnoS: BOOLEAN;              (* Indica de quién es el turno. *)
                        esperandoN, esperandoS: CARDINAL      (* Coches que esperan en cada sentido. *)
                        permisoN, permisoS: EVENT puente
                    END;

    PROCEDURE CochesNorte (i: INTEGER);
        BEGIN
            REGION puente DO BEGIN
                esperandoN := esperandoN + 1;
                WHILE ((dentroS > 0) OR (NOT turnoN & esperandoS > 0)) DO
                    AWAIT (PermisoN);
                esperandoN := esperandoN – 1;
                dentroN := dentroN + 1;
                pasadosN := (pasadosN + 1) MOD 10;      (* Nº máximo de coches en un sentido (10). *)
                IF pasadosN = 0 THEN BEGIN              (* Si es el último cambia el turno. *)
                    turnoN := FALSE;
                    turnoS := TRUE
                END
            END;
            … cruzar el puente …;
            REGION puente DO BEGIN
                dentroN := dentroN – 1;
                IF ((dentroN = 0) & (turnoS OR esperandoN = 0)) THEN
                    CAUSE (PermisoS);
            END
        END;

    PROCEDURE CochesSur (i: INTEGER);
        (* --- procedimiento simétrico a CochesNorte --- *)

    BEGIN
        WITH Puente DO BEGIN
            dentroS := 0; dentroN := 0;
            pasadosS := 0; pasadosN := 0;
            turnoN := TRUE; turnoS := TRUE;
            esperandoN := 0; esperandoS := 0
        END;
        COBEGIN
            CochesNorte (1); …; CochesNorte (n);
            CochesSur (1); …; CochesSur (m)
        COEND
    END.
```

# *El problema del Planificador*

*Se tiene un sistema con N tareas (procesos). Cada tarea debe ejecutarse (para realizar algún proceso trivial) cada cierto intervalo de tiempo, que está predefinido y almacenado como dato del programa (durante este tiempo la tarea está dormida). Se supone que el tiempo de trabajo de la tarea es menor que el de dormir. Se debe programar un planificador que arranque las tareas cuando les corresponda.*

```
PROGRAM Planificador;
    CONST N := 30;              (* Número de tareas. *)
    TYPE   tareas = [1..N];
           unaTarea =  RECORD
                           faltan: INTEGER;        (* Segundos que faltan para activar la tarea. *)
                           intervalo: INTEGER      (* Intervalo de tiempo entre activaciones. *)
                       END;
    VAR    tiempos: ARRAY tareas OF unaTarea;
           comienzo: ARRAY tareas OF SEMAPHORE;
           timer: SEMAPHORE;
           i: INTEGER

    PROCEDURE Reloj;
        CONST UNSEGUNDO := 60;        (* Ticks *)
        VAR i: INTEGER;
        BEGIN
            REPEAT
                i := UNSEGUNDO;
                WHILE i > 0 DO
                    i := i – 1;
                SIGNAL (timer)          (* Ha pasado un segundo. *)
            FOREVER
        END;

    PROCEDURE Planificador;
        VAR i := INTEGER;
        BEGIN
            REPEAT
                WAIT (timer);           (* Espera a que pase 1 segundo. *)
                FOR i:= 1 TO N DO
                    WITH tiempos[i] DO BEGIN
                        faltan := faltan – 1;
                        IF faltan = 0 THEN BEGIN
                            faltan := intervalo;          (* Activar la tarea i. *)
                            SIGNAL (comienzo[i])
                        END
                    END
                END
            FOREVER
        END;

    PROCEDURE Tarea (i: INTEGER);
        BEGIN
            REPEAT
                WAIT (comienzo[i]);
                … ejecutar el proceso i …
            FOREVER
        END;

    BEGIN
        INIT (timer, 0);
        … asignar a cada tarea su tiempo …
        FOR i := 1 TO N DO INIT (comienzo[i], 0);
        COBEGIN
            Reloj;
            Planificador;
            Tarea(1); …; Tarea (n)
        COEND
```

***END.***

# *El problema de los Lectores y Escritores*

*Los lectores pueden utilizar el recurso simultáneamente, pero cada escritor debe tener acceso exclusivo a él. Los dos tipos de procesos se excluyen mutuamente en el acceso al recurso y en caso de conflicto los escritores tienen prioridad sobre los lectores.*

## Solución con Semáforos

```
PROGRAM LectoresYEscritores;
    VAR     estado:  SHARED RECORD
                        la, lt, ea, et: INTEGER;        (* Lectores/Escritores Activos – Trabajando. *)
                     END;
             leyendo, escribiendo: SEMAPHORE

    PROCEDURE PermisoLectura;
        BEGIN
            IF ea = 0 THEN
                WHILE lt < la DO BEGIN
                    lt := lt + 1;
                    SIGNAL (leyendo)
                END
        END;

    PROCEDURE PermisoEscritura;
        BEGIN
            IF lt = 0 THEN
                WHILE et < ea DO BEGIN
                et := et + 1;
                SIGNAL (escribiendo)
            END
        END;

    PROCEDURE Lector (i: INTEGER);
        BEGIN
            REGION estado DO
                la := la + 1;
                PermisoLectura
            END;
            WAIT (leyendo);
            … acceder al recurso y leer …
            REGION estado DO BEGIN
                la := la – 1;
                lt := lt – 1;
                PermisoEscritura
            END
        END;

    PROCEDURE Escritor (i: INTEGER);
        BEGIN
            REGION estado DO BEGIN
                ea := ea + 1;
                PermisoEscritura
            END;
            WAIT (escribiendo);
            … acceder al recurso y escribir …
            REGION estado DO BEGIN
                ea := ea – 1;
                et := et – 1;
                PermisoLectura
            END
        END;
```

```
BEGIN
    INIT (leyendo, 0); INIT (escribiendo, 0);
    WITH estado DO BEGIN
        la := 0; lt := 0;
        ea := 0; et := 0
    END;
    COBEGIN
        Lector (1); …; Lector (n);
        Escritor (1); …; Escritor (m)
    COEND
END.
```

## Solución con Región Crítica Condicional

```
PROGRAM LectoresYEscritores;
    VAR     estado:  SHARED RECORD
                         lt, ea: INTEGER
                     END;

    PROCEDURE Lector (i: INTEGER);
        BEGIN
            REGION estado DO BEGIN
                AWAIT (ea = 0);
                lt := lt + 1;
            END;
            … leer …;
            REGION estado DO
                lt := lt – 1;
        END;

    PROCEDURE Escritor (i: INTEGER);
        BEGIN
            REGION estado DO BEGIN
                ea := ea + 1;
                AWAIT (lt = 0);
            END;
            … escribir …;
            REGION estado DO
                ea := ea – 1;
        END;

    BEGIN
        WITH estado DO BEGIN
            lt := 0; ea := 0
        END;
        COBEGIN
            Lector (1); …; Lector (n);
            Escritor (1); …; Escritor (m)
        COEND
    END.
```

# *El problema de Grapar Hojas*

Existen 4 montones de papeles y hay que coger uno de cada montón y grapar los cuatro juntos. El proceso se repite hasta que se acaben los montones, programar los procesos que forman los grupos de 4 papeles y otro que los tome y los grape.

## Solución con Semáforos

```
PROGRAM Grapar;
    VAR     s, aviso: SEMAPHORE;
            sigue: ARRAY [2..4] OF SEMAPHORE;
            i: INTEGER

    PROCEDURE CogerHojas1;
        VAR i: INTEGER;
        BEGIN
            REPEAT
                … tomar una hoja del montón 1 …;
                … dejar la hoja en la mesa …;
                FOR i := 2 TO 4 DO
                    WAIT (aviso);
                SIGNAL (s);
                FOR i := 2 TO 4 DO
                    SIGNAL (sigue[i]);
            UNTIL se acaben las hojas del montón 1
        END;

    PROCEDURE CogerHojasOtros (i: INTEGER);
        BEGIN
            REPEAT
                … tomar una hoja del montón i …;
                … dejar la hoja en la mesa …;
                SIGNAL (aviso);
                WAIT (sigue[i])
            UNTIL se acaben las hojas i
        END;

    PROCEDURE Grapar;
        BEGIN
            REPEAT
                WAIT (s);
                … grapar …
            UNTIL fin del proceso
        END;

    BEGIN
        INIT (s, 0); INIT (aviso, 0);
        FOR i := 2 TO 4
            INIT (sigue[i], 0);
        COBEGIN
            CojerHoja1; CojerHojasOtros(2); …; CojerHojasOtros(4)
            Grapar
        COEND
    END.
```

## Solución con Región Critica Condicional

```
PROGRAM Grapar;
    VAR     hojas: SHARED ARRAY[1..4] OF BOOLEAN;
            puede: ARRAY [1..4] OF SEMAPHORE;
            s: SEMAPHORE;
            i: INTEGER

    PROCEDURE CogerHojas (i: INTEGER);
        BEGIN
            REPEAT
                … tomar una hoja del montón i …;
                REGION hojas DO
                    hojas[i] := TRUE;
                WAIT (puede[i])
            UNTIL se acaben las hojas
        END;

    PROCEDURE Gestor;
        VAR i: INTEGER;
        BEGIN
            REPEAT
                REGION hojas DO BEGIN
                    AWAIT (hojas[1] & hojas[2] & hojas[3] & hojas[4]);
                    hojas[1] := FALSE; hojas[2] := FALSE;
                    hojas[3] := FALSE; hojas[4] := FALSE
                END;
                SIGNAL (s);
                FOR i := 1 TO 4 DO
                    SIGNAL (puede[i]);
            UNTIL se acaben los montones
        END;

    PROCEDURE Grapar;
        BEGIN
            REPEAT
                WAIT (s);
                … grapar …
            UNTIL se acabe el proceso de grapar
        END;

    BEGIN
        INIT (s, 0);
        FOR i := 1 TO 4 DO BEGIN
            hojas[i] := FALSE;
            INIT (pPuede[i], 0)
        END;
        COBEGIN
            CogerHojas(1); …; CogerHojas(4);
            Gestor; Grapar
        COEND
    END.
```

# *El problema de los Coches y el Montacargas*

En un hotel hay 10 vehículos pequeños y otros tantos grandes, controlados por un programa de procesos concurrentes (uno por vehículo). Controlar la entrada en un montacargas en el que cabe 4 coches pequeños ó 2 pequeños y 1 grande.

## Solución con Región Crítica Condicional

```
PROGRAM Montacargas;
    VAR     montacargas SHARED RECORD
                            numGrandes: 0..1;
                            numPequeños: 0..4
                        END;

    PROCEDURE CocheGrande (i: INTEGER);
        BEGIN
            REGION montacargas DO BEGIN          (* Entrada al montacargas. *)
                AWAIT (numPequeños ≤ 2) & (numGrandes = 0);
                numGrandes := numGrandes + 1
            END;
            … dentro del montacargas …
            REGION montacargas DO                (* Salida del montacargas. *)
                numGrandes := numGrandes – 1;
        END;

    PROCEDURE  CochePequeño (i: INTEGER);
        BEGIN
            REGION montacargas DO BEGIN          (* Entrada al montacargas. *)
                AWAIT ((numPequeños < 4) & (numGrandes = 0) OR
                        (numPequeños ≤ 1) & (numGrandes = 1));
                numPequeños := numPequeños + 1
            END;
            … dentro del montacargas …
            REGION montacargas DO                (* Salida del montacargas. *)
                numPequeños := NumPequeños – 1;
        END;

    BEGIN
        WITH montacargas DO BEGIN
            numPequeños := 0;
            numGrandes := 0;
        END;
        COBEGIN
            CochePequeño(1); …; CochePequeño(10);
            CocheGrande(1); …; CocheGrande(10)
        COEND
    END.
```

## Solución con Sucesos

```
PROGRAM Montacargas;
    VAR     montacargas SHARED RECORD
                                numGrandes: 0..1;
                                numPequeños: 0..4;
                                puedeGrande,
                                puedePequeño: EVENT montacargas;
                            END;

    PROCEDURE CocheGrande (i: INTEGER);
        BEGIN
            REGION montacargas DO
                WHILE NOT (numGrandes = 0) & (numPequeños ≤ 2)) DO
                    AWAIT (PuedeGrande);
                numGrandes := numGrandes + 1
            END
            … dentro del montacargas …
            REGION montacargas DO
                numGrandes := numGrandes – 1;
                CAUSE (puedeGrande);
                CAUSE (puedePequeño)
            END
        END;

    PROCEDURE CochePequeño (i: INTEGER);
        BEGIN
            REGION montacargas DO
                WHILE NOT    [ ((numGrandes = 0) & (numPequeños < 4)) OR
                               ((numPequeños ≤ 1) & (numGrandes = 1)) ] DO
                    AWAIT (puedePequeño);
                numPequeños := NumPequeños + 1
            END
            … dentro del montacargas …
            REGION montacargas DO
                numPequeños := numPequeños – 1;
                IF (numGrandes = 0) & (numPequeños ≤ 2) THEN
                    CAUSE (puedeGrande);
                END;
                CAUSE (puedePequeño)
        END;

    BEGIN
        WITH montacargas DO BEGIN
            numPequeños := 0;
            numGrandes := 0;
        END;
        COBEGIN
            CochePequeño(1); …; CochePequeño(10);
            CocheGrande(1); …; CocheGrande(10)
        COEND
    END.
```

# *El problema del Productor – Consumidor*

```
PROGRAM ProductorConsumidor
    TYPE buffer = MONITOR (capacidad: INTEGER);          (* implementa un buffer de tipo tipoBuffer *)
        VAR     contenido: ARRAY [0..MAX-1 ] OF tipoBuffer
                in, out: 0..capacidad – 1;
                cuantosHay: 0..capacidad;
                pro, con: QUEUE;

        PROCEDURE Lleno(): BOOLEAN;
            BEGIN
                RETURN cuantosHay = capacidad
            END;

        PROCEDURE Vacio(): BOOLEAN;
            BEGIN
                RETURN cuantosHay = 0
            END;

        PROCEDURE ENTRY Manda (mensaje: tipoBuffer)
            BEGIN
                IF Lleno THEN
                    DELAY (pro);
                contenido [in] := mensaje;
                in := (in + 1) MOD capacidad;
                cuantosHay := cuantosHay + 1;
                IF cuantosHay = 1 THEN
                    CONTINUE (con)
            END;

        PROCEDURE ENTRY Recibe (mensaje: tipoBuffer)
            BEGIN
                IF Vacio THEN
                    DELAY (con);
                mensaje := contenido [in];
                out := (out + 1) MOD capacidad;
                cuantosHay := cuantosHay – 1;
                IF cuantosHay = capacidad – 1 THEN
                    CONTINUE (pro)
            END;

        BEGIN           (* inicialización monitor *)
            in := 0;  out := 0; cuantosHay := 0
        END;

    VAR buzon: buffer;

    PROCEDURE Productor;
        VAR m: tipoBuffer;
        BEGIN
            REPEAT
                … elaborar mensaje …;
                buzon . Manda (m)
            UNTIL fin
        END;

    PROCEDURE Consumidor;
        VAR m: tipoBuffer;
        BEGIN
            REPEAT
                buzon . Recibe (m);
                … trabajar mensaje …
            UNTIL fin
        END;

    BEGIN
        INIT buzon(20);
        COBEGIN
            Productor;
            Consumidor
        COEND
    END.
```

# *El problema del Barbero Dormilón*

*Una barbería consiste en una habitación de espera con **n** sillas y la habitación con la silla para afeitar. Si no hay ningún cliente el barbero se va a dormir. Si un cliente entra en la barbería y ve todas las sillas ocupadas abandona la barbería. Si el barbero está ocupado, el cliente se sienta en una silla libre. Si el barbero está durmiendo el cliente lo despierta. Escríbase un programa que coordine al barbero y a los clientes.*

## Solución con Semáforos

```
PROGRAM Barbero;
    CONST  NSILLAS := 6;              (* número de sillas de la sala de espera *)
    VAR    silla, nuevoCliente, barbero: SEMAPHORE;
           esperar: INTEGER;

    PROCEDURE Cliente (i: INTEGER);
        BEGIN
            WAIT (silla);                                  (* Accede a las sillas de manera exclusiva. *)
            IF esperar < NSILLAS THEN BEGIN                (* Hay sitio en la sala de espera. *)
                esperar := esperar + 1;
                SIGNAL (nuevoCliente);      (* Despierta al barbero si es necesario. *)
                SIGNAL (silla);             (* Libera la silla. *)
                WAIT (barbero)              (* Si el barbero está ocupado, el cliente se duerme. *)
                … se corta el pelo …
            ELSE
                SIGNAL (silla);             (* Libera la silla. *)
                … pasa de largo …
            END
        END;

    PROCEDURE Barbero;
        BEGIN
            REPEAT
                WAIT (nuevoCliente);            (* Espera cliente. Si no hay clientes se duerme. *)
                WAIT (silla);                   (* Silla de la sala de espera. *)
                esperar := esperar – 1;
                SIGNAL (barbero);               (* Barbero listo para cortar el pelo. *)
                SIGNAL (silla);                 (* Activa silla. *)
                … cortar el pelo …
            FOREVER
        END;

    BEGIN
        INIT (nuevoCliente, 0);
        INIT (barbero, 0);
        INIT (silla, 1);
        COBEGIN
            Barbero;
            Cliente(1); …; Cliente(n);
        COEND
    END.
```

## Solución con Monitor

```
PROGRAM Barbero;
    TYPE barberia: MONITOR;
        VAR     sillasLibres: 0..6;
                puedeCortar: BOOLEAN:
                cliente: QUEUE;

        PROCEDURE ENTRY Silla(): BOOLEAN;
            BEGIN
                IF sillasLibres # 0 THEN BEGIN
                    sillasLibres := sillasLibres – 1;
                    RETURN TRUE
                ELSE
                    RETURN FALSE
                END
            END;

        PROCEDURE ENTRY Corte;
            BEGIN
                IF NOT puedeCortar THEN
                    DELAY(cliente);
                sillasLibres := sillasLibres + 1;
                puedeCortar := FALSE;
                … cortando …;
                puedeCortar := TRUE;
                CONTINUE(cliente)
            END;

        BEGIN
            sillasLibres := 6;
            puedeCortar := TRUE
        END;

    VAR barb: barberia;

    PROCEDURE Cliente(i: INTEGER);
        BEGIN
            … entra en barbería …;
            IF barb . Silla THEN                (* Si hay silla. *)
                barb . Corte;
            … se va de la barbería …
        END;

    BEGIN
        COBEGIN
            Cliente(1); …; Cliente(n)
        COEND
    END.
```

# *El problema del Tren Circular*

```
PROGRAM TrenCircular
    CONST E := 10                       (* E es el número de estaciones. *)
    TYPE numEstaciones: 0..E – 1;
    VAR     estación:       SHARED RECORD
                                origen, destino: EVENT ARRAY numEstaciones OF estación
                            END;
            estacionesLibres: ARRAY numEstaciones OF SEMAPHORE;
            i: INTEGER

    PROCEDURE Viajero (i: INTEGER, estaEstación, miEstación: numEstaciones);
        VAR     miEstación, estaEstación: numEstaciones;. .         .       (* ELIMINADO *)
        BEGIN
            estaEstación := n;          (* ELIMINADO *)
            miEstación := m;            (* ELIMINADO *)
            REGION estación DO BEGIN
                AWAIT (origen[estaEstación]);       (* Se duerme a la espera del tren *)
                … llega el tren …;
                … sube al tren …;                   (* Como sale de una cola es seguro que es uno a uno. *)
                AWAIT (destino[miEstación]);        (* Se duerme esperando el destino. *)
                … baja del tren …
            END
        END;

    PROCEDURE Tren (i, estoy: numEstaciones);
        VAR     estacionado: numEstaciones;         (* NO SE USA PARA NADA *)
                estoy; numEstaciones;               (* ELIMINADO *)
        BEGIN
            estoy := i;                 (* ELIMINADO *)
            SIGNAL (estoy);             (* SIN SENTIDO *)
            REPEAT
                WAIT (estacionesLibres [estoy+1 MOD E])     (* Espera a que la estación siguiente esté libre. *)
                … avanza a la siguiente estación …;
                SIGNAL (estacionesLibres [estoy MOD E])     (* Libera la estación anterior. *)
                estoy := (estoy + 1) MOD E;                 (* Llega a la siguiente estación. *)
                Tiempo (i) := 0;                            (* Se inicia el contador de tiempo. *)
                REGION estación DO BEGIN
                    CAUSE (estación . destino[estoy]);      (* Bajan viajeros. *)
                    CAUSE (estación . origen [estoy])       (* Suben viajeros. *)
                END;
                IF Tiempo (i) < 120 THEN                    (* Si no han pasado 120 segundos espera. *)
                    … esperar …;
            FOREVER
        END;

    BEGIN
        FOR i := 0 TO numEstaciones DO
            INIT (estacionesLibres[i], 0);      (* Estos semáforos se deberían inicializar coherentemente con *)
        COBEGIN                                 (* el punto de inicio de cada tren. *)
            Viajero (1, m, n); …        (* m y n son las estaciones origen y destino. *)
            Tren (1, p); …              (* p es la estación inicial de cada tren. *)
        COEND
    END.
```

# *El problema del Cine*

*Sea un cine con tres salas (aforos de 100, 100 y 200 espectadores) y una única taquilla de entradas. Los espectadores llegan a la taquilla y, al azar, escogen una sala. Si no hay entradas se van. Si las hay entran en la sala. No está previsto que abandonen las sala una vez iniciada la sesión.*

```
PROGRAM Cine;
    VAR taquilla: SHARED ARRAY [1..3] OF INTEGER;

    PROCEDURE Espectador (i: INTEGER; sala: 1..3)
        VAR     entrada: BOOLEAN
        BEGIN
            entrada := FALSE;
            REGION taquilla DO
                IF taquilla[sala] > 0 THEN BEGIN
                    taquilla[sala] := taquilla[sala] – 1;
                    entrada := TRUE
                END
            END;
            IF entrada THEN
                … ver película …;
            … irse del cine …
        END;

    BEGIN
        taquilla[1] := 100; taquilla[2] := 100; taquilla[3] := 200;
        COBEGIN
            Espectador(1, s1); … Espectador(n, sm)          (* sm indica la sala que el espectador elige. *)
        COEND
    END.
```

# *El problema de los Niños*

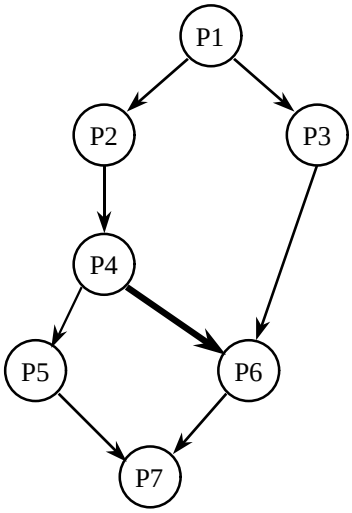
*Seis niños siguen repetitivamente la siguiente secuencia de acciones: pintan, cuando se cansan se lavan, comen 1 de los 4 platos que hay y duermen.*
*Existe un proceso cuidador que repone el jabón para lavarse (hasta 50 veces) y pone comida en los platos.*

```
PROGRAM Niños;
    VAR     jabón: SHARED INTEGER;
            comida: SHARED RECORD
                        platosMesa,
                        platosSucios: INTEGER;
                    END;
            dosis: INTEGER;

    PROCEDURE Niño(i: INTEGER);
        BEGIN
            REPEAT
                REGION jabón DO BEGIN
                    AWAIT jabón = 1;
                    jabón := 0
                END;
                … se ha lavado …;
                REGION comida DO BEGIN
                    AWAIT platosMesa > 0;
                    platosMesa := platosMesa – 1
                END;
                … come …;
                REGION comida DO
                    platosSucios := platosSucios + 1;
            FOREVER
        END;

    PROCEDURE Cuidador;
        BEGIN
            WHILE dosis > 0 DO BEGIN
                REGION jabón DO BEGIN
                    IF jabón = 0 THEN BEGIN        (* Repone jabón si es necesario. *)
                        jabón := 1;
                        dosis := dosis – 1
                    END
                END;
                REGION comida DO BEGIN
                    WHILE platosSucios > 0 DO BEGIN          (* Repone platos si es necesario. *)
                        platosSucios := platosSucios – 1;
                        platosMesa := platosMesa +1
                    END
                END
            END
        END;

    BEGIN
        dosis := 50;
        jabón := 1;
        WITH comida DO
            platosMesa := 4;
            platosSucios := 0
        END;
        COBEGIN
            Niño(1); …; Niño(6);
            Cuidador
        COEND
    END.
```

# *Codificación de ejecución de procesos (secuencial y paralela)*

## Examen 28 Enero 1997 y 15 Septiembre 1998

```
PROGRAM Ejecución;

    VAR s: SEMAPHORE;

    PROCEDURE P1;
        BEGIN
            ……
        END;

    ……

    PROCEDURE P7;
        BEGIN
            ……
        END;

BEGIN
    P1;
    COBEGIN
        BEGIN
            P2;
            P4;
            SIGNAL(s);
            P5
        END;
        BEGIN
            P3;
            WAIT(s);
            P6
        END
    COEND;
    P7
END.
```




## Examen 11 Febrero 1997

```
PROGRAM Ejecución;

    VAR s: SEMAPHORE;

    PROCEDURE P1;
        BEGIN
            ……
        END;

    ……

    PROCEDURE P8;
        BEGIN
            ……
        END;

BEGIN
    P1;
    COBEGIN
        BEGIN
            P2;
            COBEGIN
                P3;
                P4
            COEND;
            SIGNAL(s);
            P5;
        END;
        BEGIN
            P6;
            WAIT(s);
            P7
        END
    COEND;
    P8
END.
```