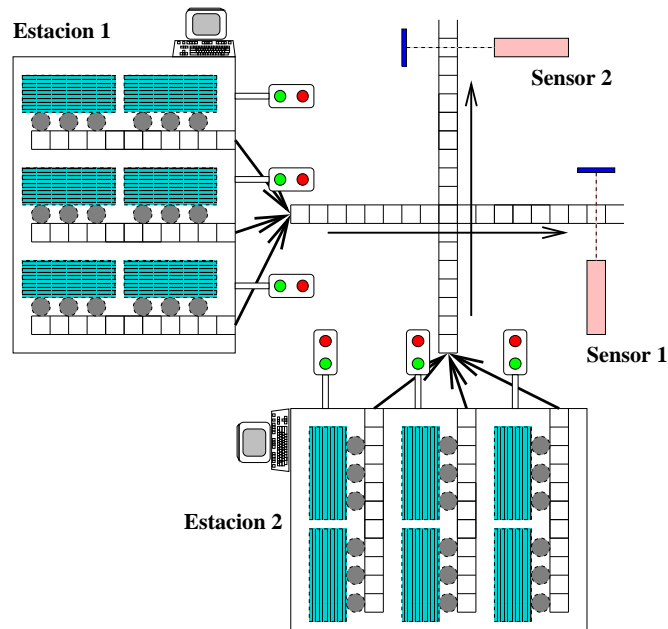


1 Diseño de un controlador para un cruce de trenes

[4 puntos]

Se pretende diseñar un controlador que regule las salidas de dos estaciones de tren cuyas vías se cruzan. El aspecto de las estaciones y las vías se muestra a continuación:



En cada estación se cargan trenes con mercancías y se solicita la salida de un tren cuando está lleno. La forma de solicitar la salida es mediante un terminal de ordenador en que se introduce el número de vía en la que está estacionado el tren y qué tipo de tren es (ver más abajo). Un tren arranca cuando su semáforo se pone en verde. Tras salir de la estación el tren entra en el cruce, en el que no debe haber dos trenes simultáneamente. A la salida del cruce hay un sensor en cada vía que detecta cuándo el tren ya ha abandonado el cruce.

Hay trenes de dos tipos: urgentes y normales. Dentro de cada estación, los trenes urgentes tienen siempre preferencia sobre los normales y la salida de estos últimos debe retrasarse hasta que no haya ningún tren urgente esperando. Dentro de la misma categoría de trenes y en una estación dada las peticiones de salida deben ser atendidas en el mismo orden en que fueron realizadas.

Tenemos el siguiente interfaz para leer datos acerca de los sensores de las vías y de las peticiones de salida de trenes, y para notificar la salida de los trenes:

```
Sensor . EsperarPasoTren(NumeroEstacion: TipoNumEst)
(* Se bloquea hasta que se detecte el paso del último vagón de un tren *)
(* por la vía que sale de la estación NumeroEstacion. *)
(* Puede consultarse concurrentemente el estado de varios sensores. *)
```

```
Semaforo . DarSalida(NumeroEstacion: TipoNumEst;
                     NumeroVia: TipoNumVia);
(* Pone a verde el semáforo de la vía NumeroVia en la estación *)
(* NumeroEstacion y lo vuelve a poner a rojo tras el tiempo necesario *)
(* para que el tren arranque (este tiempo es mucho menor que el que *)
(* emplea el tren en llegar al cruce). No pueden realizarse *)
(* concurrentemente operaciones sobre semáforos de la misma estación. *)
(* Es bloqueante. *)
```

```
Terminal . LeerPeticion( NumeroEstacion: TipoNumEst;
                        VAR Clase: TipoTren;
                        VAR NumeroVia: TipoNumVia);
(* Lee una petición de salida de la estación NumeroEstacion. La vía *)
(* en la que está el tren que quiere salir se devuelve en NumeroVia *)
(* y la clase de tren se devuelve en Clase. Es bloqueante. *)
```

Se suponen las definiciones de tipos

```

TYPE TipoNumEst = [1..2];
        TipoNumVia = 1..MaxVias;
        TipoTren   = (urgente, normal);

```

y se asume que no hay que comprobar la corrección de los datos que llegan del terminal (por ejemplo, nunca se va a decir que hay un tren esperando en una vía cuando en realidad no lo hay) ni preocuparse por eventualidades ajenas al funcionamiento correcto de la estación (por ejemplo, que un tren no pueda arrancar).

Se pide: realizar un diseño para el controlador del cruce que garantice que no habrá choques y que respete las prioridades antedichas entre trenes. Se sugiere seguir los siguientes pasos:

1. Dar el esquema de recursos y procesos. Incluir los nombres de las operaciones en el/los recurso(s) y señalar claramente qué operaciones usa cada proceso.
2. Redactar el código de los procesos que aparezcan en el punto anterior. La comunicación con el personal de la estación y con el sensor del cruce debe utilizar únicamente el interfaz expuesto anteriormente.
3. Especificar formalmente con un C-TAD el/los recurso(s) que aparezcan en el primer apartado de esta pregunta.

Nota: aunque el texto del problema habla únicamente de dos estaciones, pensar en el problema como si tuviese n estaciones puede ayudar a tener una solución más elegante.

Solución propuesta:

Una posible solución se encuentra al principio del ejercicio 2. Otra posible variante, igualmente válida, utiliza únicamente dos clases de procesos: una de ellas inserta en el recurso compartido las peticiones del terminal, y el otro recupera la siguiente petición a ser satisfecha y espera a que pase por el cruce:

<pre> TASK AdmisionPetición(i) LOOP Terminal.LeerPetición(i, ClaseTren, Via); Estacion.TrenListo(i, ClaseTren, Via); END; END; </pre>	<pre> TASK ControladorCruce; LOOP Estacion.PermisoSalirEst(i, Via); Semaforo.DarSalida(i, Via); Sensor.EsperarPasoTren(i); END; END; </pre>
---	---

Esta solución también simplifica el recurso compartido, pues Estacion.PermisoSalirEst no tiene que tener en cuenta la ocupación del cruce. La solución se basa fuertemente en la cadencia y no cambio de vía de los trenes que salen de las estaciones.

Entre los **errores** cometidos se encuentran:

- Tener dos recursos, uno por estación. Esto obliga a actualizar en ambos el estado del cruce. Aún en el caso de que esto se haga, uno de los recursos se actualizará antes que el otro, lo que puede dar lugar a una preferencia implícita.
- Un recurso para estación y otro para el cruce tiene un problema similar: es necesario llevar la información del cruce al recurso que modela la estación.
- Leer una petición y esperar a que se den las condiciones adecuadas para atender esa petición:

```

TASK AdmisionPetición(i)
  LOOP
    Terminal.LeerPetición(i, ClaseTren, Via);
    Estacion.TrenListo(i, ClaseTren, Via);
    Estacion.PermisoSalirEst(i, Via);
    Semaforo.DarSalida(i, Via);
    Sensor.EsperarPasoTren(i);
  END;
END;

```

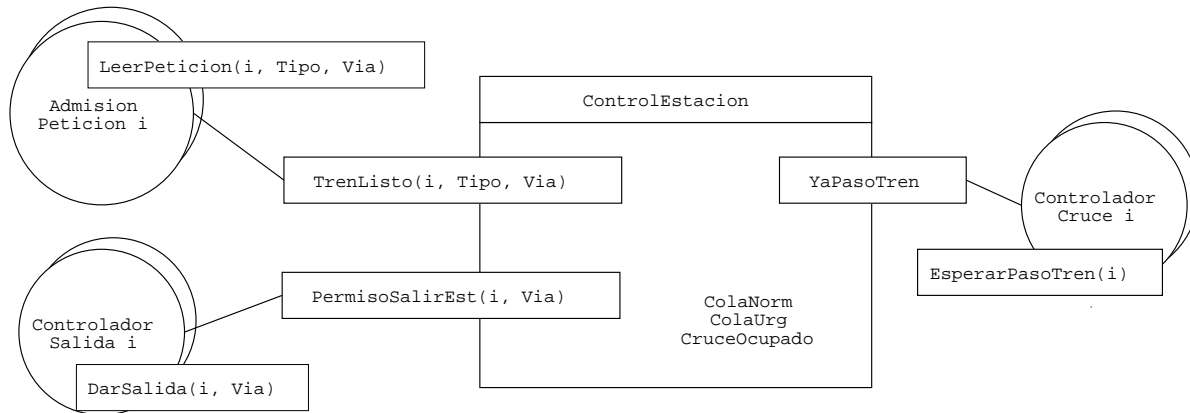
Hay dos problemas en esta solución: el primero es que la aceptación de peticiones en el terminal se ve retrasada hasta que salga el tren anterior (porque el proceso está bloqueado en `Estacion.TrenListo(i, ClaseTren, Via);`), con lo que se podrían generar colas de empleados enfadados porque aunque el tren está cargado, el terminal no responde.

Podría suponerse que el terminal tiene cierta inteligencia y “encola” las peticiones que le entran. Esto causa un problema adicional: en el tiempo entre que se lee una petición y se atiende, pueden haberse llenado otros trenes de mayor prioridad. Desde el punto de vista de la estación, ello significa que habiendo trenes urgentes listos para salir, están saliendo otros normales, sencillamente porque se han introducido antes sus datos.

- No diferenciar trenes urgentes y trenes normales.
- No guardar ninguna referencia a la vía en la que está un tren.
- No guardar ninguna cola de peticiones.
- Hacer llamadas bloqueantes dentro del recurso (esto **paraliza** el recurso durante un tiempo *a priori* no determinado).
- Asignar un proceso por tren: esto es una simulación que **no** es lo que se pedía en el enunciado. Implica, además, el acceso concurrente a los terminales de la estación. Aunque esto no está explícitamente prohibido en el enunciado, no suele ser buena idea. En muchos casos lleva a un uso incorrecto de las llamadas `Terminal.LeerPeticion(i, ClaseTren, Via)`, en la que `Via` aparece como parámetro de entrada y no de salida.

2 Implementación de un controlador para un cruce de trenes [4 puntos]

Se ha llegado al siguiente diseño (aunque hay otros posibles, igualmente válidos) para el problema planteado en la pregunta 1:



Hay un proceso $\text{AdmisionPeticion}(i)$ por cada estación que se encarga de esperar a que un tren esté cargado y transmite esta información al recurso. El proceso $\text{ControladorSalida}(i)$ permite la salida de un tren en la estación i poniendo el semáforo correspondiente a verde. El proceso $\text{ControladorCruce}(i)$ señala que el cruce está libre porque ya ha pasado el tren. El pseudocódigo de los procesos es:

```

TASK AdmisionPeticion(i)
  LOOP
    Terminal.LeerPeticion(i, ClaseTren, Via);
    Estacion.TrenListo(i, ClaseTren, Via);
  END;
END;

TASK ControladorSalida(i)
  LOOP
    Estacion.PermisoSalirEst(i, Via);
    Semaforo.DarSalida(i, Via);
  END;
END;

TASK ControladorCruce(i)
  LOOP
    Sensor.EsperarPasoTren(i);
    Estacion.YaPasoTren;
  END;
END;

```

La especificación del C-TAD es la siguiente:

C-TAD *ControlEstacion*

C-Operaciones

ACCION Inicializar: *TipoControlEstacion*

ACCION TrenListo: *TipoControlEstacion* \times *TipoNumEst* \times *TipoTren* \times *TipoVia*

ACCION PermisoSalirEst: *TipoControlEstacion* \times *TipoNumEst* \times *TipoVia*

ACCION YaPasoTren: *TipoControlEstacion*

TRANSACCIONES

Inicializar

TrenListo

PermisoSalirEst

YaPasoTren

CONCURRENCIA

ninguna

DOMINIO

Tipo: *TipoControlEstacion* = (*CruceOcupado*: booleano \times
ColaUrg: Secuencia(Secuencia(*TipoVia*)) \times
ColaNorm: Secuencia(Secuencia(*TipoVia*)))

TipoVia = 1..MaxVias

TipoNumEst = 1..2

TipoTren = Urgente | Normal

Invariante: $\forall t \in \text{TipoControlEstacion} \bullet (\text{Longitud}(t.\text{ColaUrg}) = 2 \wedge \text{Longitud}(t.\text{ColaNorm}) = 2)$

CPRE: cierto

Inicializar(Cnt)

POST: Prepara el estado inicial del recurso

POST: $Cnt.CruceOcupado^{sal} = \text{falso} \wedge$

$\text{Longitud}(Cnt.ColaUrg^{sal}(1)) = 0 \wedge \text{Longitud}(Cnt.ColaUrg^{sal}(2)) = 0 \wedge$

$\text{Longitud}(Cnt.ColaNorm^{sal}(1)) = 0 \wedge \text{Longitud}(Cnt.ColaNorm^{sal}(2)) = 0$

CPRE: cierto

TrenListo(Cnt, Est, Tipo, Via)

POST: Indica que un tren se encuentra listo para salir.

Se guarda la vía en la cola correspondiente a su tipo y estación

POST: $Cnt^{sal} = Cnt^{ent} \setminus (Tipo = Urgente \rightarrow Insertar(Cnt.ColaUrg(Est), Via)) \wedge$

$(Tipo = Normal \rightarrow Insertar(Cnt.ColaNorm(Est), Via))$

Donde: $Insertar(s, e) = (s^{sal} = s^{ent} \setminus \text{Longitud}(s^{sal}) = \text{Longitud}(s^{ent}) + 1 \wedge s^{sal}(\text{Longitud}(s^{sal})) = e)$

CPRE: $(HayUrgentes \vee HayNormales) \wedge \neg Cnt.CruceOcupado^{ent}$

PermisoSalirEst(Cnt, Est, Via)

POST: Cuando hay trenes esperando a salir y el cruce no está ocupado, se desbloquea y devuelve el numero de vía en la que está el tren con más alta prioridad

POST: $Cnt^{sal} = Cnt^{ent} \setminus Cnt.CruceOcupado^{sal} = \text{verdad} \wedge (HayUrgentes \rightarrow Retirar(Cnt.ColaUrg(Est), Via))$

$(\neg HayUrgentes \wedge HayNormales \rightarrow Retirar(Cnt.ColaNormal(Est), Via))$

Donde: $LongUrg = \text{Longitud}(Cnt.ColaUrg^{ent}(Est)) \wedge LongNorm = \text{Longitud}(Cnt.ColaNormal^{ent}(Est)) \wedge$

$HayUrgentes = (LongUrg > 0) \wedge HayNormales = (LongNorm > 0) \wedge$

$Retirar(s, e) = (e^{sal} = s^{ent}(1) \wedge l = \text{Longitud}(s^{ent}) \wedge \text{Longitud}(s^{sal}) = l - 1 \wedge s^{sal}(1..l - 1) = s^{ent}(2..l))$

CPRE: cierto

YaPasoTren(Cnt)

POST: Indica que el cruce se encuentra libre

POST: $Cnt^{sal} = Cnt^{ent} \setminus Cnt.CruceOcupado^{sal} = \text{falso}$

Se pide: basándose en la especificación anterior, codificar, utilizando el lenguaje CcModula y mediante la técnica de paso de mensajes síncrono y la metodología explicada en clase, un recurso activo que implemente las operaciones del C-TAD *ControlEstacion*. Pueden suponerse implementaciones ya realizadas (pero no necesariamente preparadas para un acceso concurrente) de los tipos de datos abstractos básicos (listas, colas, pilas, etc.). La implementación debe evitar la inanición entre estaciones: un tren que en una estación puede salir inmediatamente porque es el primero teniendo en cuenta su tipo y posición en su cola de salida, debe recibir permiso para salir en un tiempo finito.

Solución propuesta:

```
CONST
    NUMESTACIONES = 2;
    MAXVIAS = 2;

(* DECLARACION DE TIPOS *)

TYPE
    TipoVacio = CARDINAL;
    TipoNumEst = [1..NUMESTACIONES];
    TipoVia = CARDINAL;
    TipoTren = (urgente, normal);
    TipoSalida = RECORD
        numestacion: TipoNumEst;
        numvia: TipoVia;
        tipotren: TipoTren;
    END;
```

```

VAR
    (* DECLARACION DE LOS CANALES *)
    canTrenListo: CHANNEL; (* OF TipoSalida *)
    canPermisoSalirEst: ARRAY [1..NUMESTACIONES] OF CHANNEL; (* OF TipoVacio *)
    canYaPasoTren: CHANNEL; (* OF TipoVacio *)
    canDevVia: ARRAY [1..NUMESTACIONES] OF CHANNEL; (* OF TipoVia *)

(* CODIGO QUE IMPLEMENTA EL RECURSO ACTIVO *)

TASK ControlEstacion;
VAR
    (* Estado del recurso *)
    cruceocupado: BOOLEAN;
    colaurg, colanorm: ARRAY TipoNumEst OF COLA.TipoCola;
    (* variables auxiliares *)
    i: TipoNumEst;
    viaquesale: TipoVia;
    datossalida: TipoSalida;
    vacio: TipoVacio;

BEGIN
    (* Inicializacion del recurso *)

    GetChannel (canTrenListo);
    GetChannel (canYaPasoTren);
    FOR i := 1 TO NUMESTACIONES DO
        GetChannel (canPermisoSalirEst[i]);
        COLA.Vacia(colaurg[i]);
        COLA.Vacia(colanorm[i]);
    END;
    cruceocupado := FALSE;

    LOOP
        SELECT
            (* Canal para la operacion YaPasoTren *)

            WHEN cruceocupado, Receive (canYaPasoTren, vacio) DO
                cruceocupado := FALSE;

            (*
                Para la operacion PermisoSalirEst se emplean dos canales,
                uno para cada estacion (1 y 2). Dado que CCMODULA elige de
                forma equitativa entre todos los canales abiertos en una
                espera selectiva, se asegura la equidad entre las dos estaciones,
                y por tanto se evita la inanici?n de una de las estaciones.
            *)

            WHEN NOT (cruceocupado) AND (NOT (COLA.EsVacia(colaurg[1])) OR
            NOT (COLA.EsVacia(colanorm[1]))),
                Receive (canPermisoSalirEst[1], vacio) DO
                    cruceocupado := TRUE;

            (* si hay trenes urgentes esperando en esta estacion,
                deben salir antes *)
            IF NOT(COLA.EsVacia(colaurg[1])) THEN
                COLA.Primerio(colaurg[1], viaquesale);
                COLA.Borrar(colaurg[1]);
            ELSE
                COLA.Primerio(colanorm[1], viaquesale);

```

```

        COLA.Borrar(colanorm[1]);
    END;

    (* se devuelve la via del tren que va a salir desde
       la estacion 1 *)
    Send (canDevVia[1], viaquesale);

    WHEN NOT (cruceocupado) AND (NOT (COLA.EsVacia(colaurg[2])) OR
    NOT (COLA.EsVacia(colanorm[2]))),
        Receive (canPermisoSalirEst[2], vacio) DO
        cruceocupado := TRUE;

        (* si hay trenes urgentes esperando en esta estacion,
           deben salir antes *)
        IF NOT(COLA.EsVacia(colaurg[2])) THEN
            COLA.Primerio(colaurg[2], viaquesale);
            COLA.Borrar(colaurg[2]);
        ELSE
            COLA.Primerio(colanorm[2], viaquesale);
            COLA.Borrar(colanorm[2]);
        END (* IF *);
        (* se devuelve la via del tren que va a salir desde
           la estacion 2 *)
        Send (canDevVia[2], viaquesale);

    (* Canal para la operacion TrenListo *)

    WHEN TRUE, Receive (canTrenListo, datossalida) DO
        IF datossalida.tipotren = urgente THEN
            COLA.Insertar(colaurg[datossalida.numestacion], datossalida.numvia);
        ELSE
            COLA.Insertar(colanorm[datossalida.numestacion], datossalida.numvia);
        END;

    END (* SELECT *)
    END (* LOOP *)
END ControlEstacion;

(*Codigo de las operaciones del recurso *)

PROCEDURE TrenListo (i: TipoNumEst; clasetren: TipoTren; via: TipoVia);
VAR
    datossalida: TipoSalida;
BEGIN
    datossalida.numestacion := i;
    datossalida.numvia := via;
    datossalida.tipotren := clasetren;
    Send (canTrenListo, datossalida);
END TrenListo;

PROCEDURE YaPasoTren;
VAR
    vacio: CARDINAL;
BEGIN
    Send (canYaPasoTren, vacio); (* se manda una sennal *)
END YaPasoTren;

```

```
PROCEDURE PermisoSalirEstacion (i: TipoNumEst; VAR via: TipoVia);  
VAR  
    vacio: CARDINAL;  
BEGIN  
    Send (canPermisoSalirEst[i], vacio);  
    Receive (canDevVia[i], via);  
END PermisoSalirEstacion;
```