

Examen de Programación Concurrente

febrero 2004

Dpto. LSIIS. Unidad de Programación.

Normas

Esta prueba consta de dos partes diferenciadas. La primera, cuyo enunciado estás leyendo en este momento, contiene la primera parte y se recogerá **una hora y media** después de haberse entregado. Después se entregará el enunciado de la segunda parte. La puntuación total del examen es de **10 puntos**.

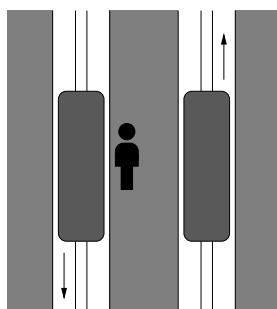
Las soluciones aparecerán publicadas antes de la fecha de revisión y las calificaciones se darán a conocer el **17 de febrero de 2004**. La revisión de este examen tendrá lugar el **19 de febrero de 2004**.

1. People mover

[1 hora 30 min., 5 puntos]

Se da el nombre de *people mover* a aquellos sistemas de ferrocarril ligero o monorraíl carentes de conductor y que suelen usarse para salvar pequeñas distancias, p.ej. las distintas terminales de un aeropuerto o los edificios de un campus.

El sistema que vamos a considerar consiste en una línea circular con N estaciones y 2 vías, una en cada sentido. Por cada vía viaja un único tren. La figura muestra una vista aérea de una estación.



Existe un andén central, de entrada, por el que se puede acceder indistintamente a los trenes de ambas vías. En la figura, un pasajero se dispone a montar en uno de los trenes. La salida ha de realizarse necesariamente por los andenes exteriores. Para ello los trenes disponen de puertas de entrada y salida que se accionan de manera independiente.

Para ahorrar energía, los trenes se encuentran detenidos en alguna estación hasta que un pasajero se monte o hasta que algún pasajero de otra estación lo solicite mediante el botón de llamada presente en todos los andenes centrales.

Cada tren dispone de un sensor capaz de detectar que se está aproximando a una estación, pero sólo se debe detener si es necesario, es decir, si uno de los pasajeros que viajan en el tren ha pulsado el botón de *parada solicitada* o si se ha llamado al tren desde la estación a la que éste se acerca. La solicitud de parada se desactiva al detenerse el tren y no puede volverse a activar hasta que se reanuda la marcha.

El tren posee un detector de carga que le permite conocer si lleva pasajeros en un momento dado. Esto sirve, entre otras cosas, para detectar que un pasajero ha subido a un tren vacío y detenido en una estación.

Para realizar el control de este sistema se dispone de los siguientes procedimientos **ya programados**:

```
procedure Ocupacion_Vagon(in  T: Tipo_Tren;
                           out P: Boolean);
```

Se encarga de ver si la carga del tren T permite deducir que hay algún pasajero. Esta operación es relativamente lenta (tarda unos segundos).

```
procedure Lectura_Anden(in  E: Tipo_Estacion);
```

Se queda bloqueado hasta que se detecta la llamada desde una estación.

```
procedure Lectura_Parada(in  T: Tipo_Tren);
```

Se queda bloqueado hasta que se detecta la petición de parar en la próxima estación.

```
procedure Detector(in T: Tipo_Tren);
```

Se queda bloqueado hasta que se detecta la proximidad a una estación.

```
procedure Arrancar_Tren(in  T: Tipo_Tren);
procedure Detener_Tren (in  T: Tipo_Tren);
procedure Apertura_Puertas_Salida(in  T: Tipo_Tren);
procedure Cierre_Puertas_Salida  (in  T: Tipo_Tren);
procedure Apertura_Puertas_Entrada(in  T: Tipo_Tren);
procedure Cierre_Puertas_Entrada  (in  T: Tipo_Tren);
```

Actúan sobre diferentes dispositivos del tren. Deben ser llamados de forma no simultánea para un tren dado y en una secuencia coherente con los requisitos planteados anteriormente. Una llamada a una de estas operaciones puede tardar unos segundos en retornar: p.ej., *Cierre_Puertas_Salida(t)* incluye todo el proceso de cerrado, con aviso sonoro, pequeña espera y cierre de las puertas en sí. Para simplificar, supondremos que el intento de cerrar unas puertas ya cerradas o abrir unas puertas ya abiertas carece de efecto alguno.

Se pide:

- Grafo de procesos/recursos que permita controlar el sistema arriba descrito.
- Código esquemático de las tareas propuestas.
- Especificación formal (CTADSOL) del recurso o recursos propuestos.

Examen de Programación Concurrente

febrero 2004

Dpto. LSIS. Unidad de Programación.

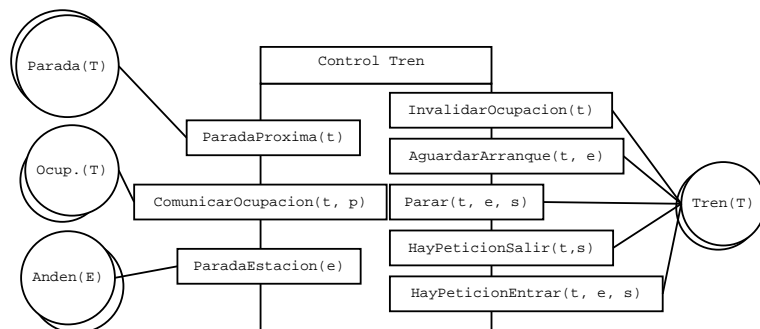
Normas

Disponéis de **una hora y media** para responder a los apartados de esta parte. Cada uno de ellos se deberá entregar en hojas separadas.

2. Implementación

[1 hora 30 min., 5 puntos]

Tras un estudio se ha llegado a un diseño del que mostramos el gráfico de procesos y recursos (a la derecha), el pseudocódigo de los diferentes procesos y la especificación del recurso compartido (a continuación).



Ocupacion(T: in Tipo_Tren): loop — <i>Recalcular ocupacion</i> Ocupacion_Vagon(T,P); R. Comunicar_Ocupacion(T,P); end loop ;	Anden(E: in T_Estacion): loop Lectura_Anden(E); R. Parada_Estacion(E); end loop ;	Parada(T: in Tipo_Tren): loop Lectura_Parada(T); R. Parada_Proxima(T); end loop ;
---	--	--

```

Tren(T: in Tipo_Tren):

    Estacion_Actual := ...           — Cualquiera
    Apertura_Puertas_Entrada(T);     — Inicio de jornada: abrir puertas

    loop
        Cierre_Puertas_Salida(T);     — e iniciar marcha del tren
        R.Invalidar_Ocupacion(T);
        R.Aguardar_Arranque(T, Estacion_Actual);
        Cierre_Puertas_Entrada(T);    — Cerrar puertas tras admitir viajeros
        Arrancar_Tren(T);

        loop — Viajar hasta llegar a estación con peticiones
            Detector(T); — Bloquea hasta llegada siguiente estación
            Estacion_Actual := Siguiente(Estacion_Actual);
            R.Parar(T, Estacion_Actual, Pasajeros);
        exit when Pasajeros;
        end loop
        Detener_Tren(T); — Van a entrar o salir
                        — Examinamos peticiones
        R.Hay_Peticion_Salir(T, Salir); — Sólo consultamos
        if Salir then
            Apertura_Puertas_Salida(T);
        end if;

        R.Hay_Peticion_Entrar(Estacion_Actual, Entrar); — Sólo consultamos
        if Entrar then
            Apertura_Puertas_Entrada(T);
        end if;
    end loop;

```

C-TADSOL ControlTren**OPERACIONES**

ACCIÓN ComunicarOcupacion: $ControlTren[es] \times TipoTren[e] \times \mathbb{B}[e]$

ACCIÓN InvalidarOcupacion: $ControlTren[es] \times TipoTren[e]$

ACCIÓN ParadaEstacion: $ControlTren[es] \times TipoEstacion[e]$

ACCIÓN ParadaProxima: $ControlTren[es] \times TipoTren[e]$

ACCIÓN HayPeticionSalir: $ControlTren[es] \times TipoTren[e] \times \mathbb{B}[s]$

ACCIÓN HayPeticionEntrar: $ControlTren[es] \times TipoEstacion[e] \times \mathbb{B}[s]$

ACCIÓN Parar: $ControlTren[es] \times TipoTren[e] \times TipoEstacion[e] \times \mathbb{B}[s]$

ACCIÓN AguardarArranque: $ControlTren[es] \times TipoTren[e] \times TipoEstacion[e]$

SEMÁNTICA**DOMINIO:**

TIPO: $ControlTren = (pet_est: Secuencia(\mathbb{B}) \times pet_tren: Secuencia(\mathbb{B}) \times$
 $ocupado: Secuencia(\mathbb{B}) \times valido: Secuencia(\mathbb{B}))$

$TipoEstacion = 1..NEstaciones$

$TipoTren = 1..NTrenes$

INVARIANTE: $\forall t \in ControlTren \bullet Longitud(t.ocupado) = NTrenes \wedge Longitud(t.valido) =$
 $NTrenes \wedge Longitud(t.pet_tren) = NTrenes \wedge Longitud(t.pet_est) =$
 $NEstaciones$

INICIAL(t): $(\forall i, 1 \leq i \leq NTrenes \bullet \neg t.ocupado(i) \wedge \neg t.valido(i) \wedge \neg t.pet_tren(i)) \wedge$
 $(\forall i, 1 \leq i \leq NEstaciones \bullet \neg t.pet_est(i))$

CPRE: Cierto

ComunicarOcupacion(r, t, o)

POST: $r^{sal} = r^{ent} \setminus r^{sal}.ocupado(t) = o \wedge r^{sal}.valido(t)$

CPRE: Cierto

InvalidarOcupacion(r, t)

POST: $r^{sal} = r^{ent} \setminus \neg r^{sal}.valido(t)$

CPRE: Cierto

ParadaEstacion(r, e)

POST: $r^{sal} = r^{ent} \setminus r^{sal}.pet_est(e)$

CPRE: Cierto

ParadaProxima(r, t)

POST: $r^{sal} = r^{ent} \setminus r^{sal}.pet_tren(t)$

CPRE: Cierto

HayPeticionSalir(r, t, s)

POST: $r^{sal} = r^{ent} \setminus \neg r^{sal}.pet_tren(t) \wedge s^{sal} = r^{ent}.pet_tren(t)$

CPRE: Cierto

HayPeticionEntrar(r, e, s)

POST: $r^{sal} = r^{ent} \setminus \neg r^{sal}.pet_est(e) \wedge s^{sal} = r^{ent}.pet_est(e)$

CPRE: Cierto

Parar(r, t, e, s)

POST: $r^{ent} = r^{sal} \wedge s^{sal} = (r.pet_tren(t) \vee r.pet_est(e))$

CPRE: $(r.ocupado(t) \wedge r.valido(t)) \vee \exists i \in TipoEstacion \bullet i \neq e \wedge r.pet_est(i)$

AguardarArranque(r, t, e)

POST: $r^{ent} = r^{sal}$

Se pide:

- Una implementación (parcial) en Ada 95 del **C-TADSOL** utilizando objetos protegidos que incluya la definición de tipos y variables de estado necesarias y el código de las operaciones **AguardarArranque**, **HayPeticiónEntrar** y **ParadaEstacion**. [2 puntos]
- Una implementación (parcial) en Ada 95 del **C-TADSOL** utilizando *Rendez-Vous* (con paso de mensajes explícito si se considera necesario) que incluya la definición de tipos y variables de estado necesarias y el código de las operaciones **AguardarArranque**, **HayPeticiónEntrar** y **ParadaEstacion**. [2 puntos]
- La comunicación de la ocupación (o no) de un vagón sólo incumbe a los procesos controlador del tren y al proceso que recoge dicha información. ¿Podría extraerse esa comunicación del recurso presentado y delegarse a otro recurso? Razonar **breve y claramente** la respuesta. [1 punto]