

Departamento de Lenguajes, Sistemas Informáticos e Ingeniería del Software

Este examen consta de tres apartados: un planteamiento de un problema con una solución incompleta, un cuestionario y una pregunta de implementación. La duración total es de **tres horas**. El cuestionario debe contestarse **en la misma hoja** en que se os entrega (consultad con un profesor si necesitáis otra hoja) y se podrá devolver hasta **una hora** después de recibirlo. Al hacerlo se os dará la pregunta de implementación.

La puntuación total del examen es de **10 puntos**. Las soluciones se proporcionarán antes de la revisión. Las calificaciones se darán a conocer el día **20 de junio**. La revisión del examen tendrá lugar el día **22 de junio**.

1. Avisador de correo

Se desea programar un avisador de correo: un pequeño icono en el ordenador que debe alertar ante la llegada de correo. Los mensajes que llegan se depositan, de modo automático, en un almacén que puede ser un fichero local o remoto (no es relevante) al que podemos acceder para saber su tamaño y detectar si ha habido cambios en el mismo.

El funcionamiento esperado es el siguiente: cuando no hay correo pendiente el icono del avisador debe estar apagado. Al llegar correo debe iluminarse. Si se lee el correo o se pulsa sobre el icono con el ratón, éste debe desiluminarse. En este segundo caso no se iluminaría de nuevo, aunque llegasen nuevos mensajes, hasta que no se realizase la lectura del correo pendiente. Si transcurren más de L segundos tras la llegada del primer correo a un buzón vacío sin que el correo se lea o se pulse sobre el icono con el ratón, el avisador deberá parpadear (iluminarse y desiluminarse alternativamente). Si se pulsa el ratón sobre el avisador en este estado, debe desiluminarse y permanecer así hasta que se lea el correo pendiente y llegue más.

Las primitivas del sistema (inamovibles) de las que disponemos son las siguientes:

procedure Esperar.Hasta(T : in Tipo.Tiempo);

— *Suspende el proceso llamante hasta el tiempo absoluto T (por ejemplo, hasta las 15:21:05). Tipo.Tiempo es un TAD que admite comparaciones, sumas, restas, etc.*

procedure Hora(T : out Tipo.Tiempo);

— *Devuelve la hora actual. Regresa inmediatamente.*

function Tam.Correo return Natural;

— *Devuelve el número de mensajes en el fichero de correo. Regresa inmediatamente.*

function Cambio.Correo return Natural;

— *Se bloquea hasta que el fichero de correo cambie. Devuelve el número de mensajes*

procedure Espera.Raton;

— *Se queda bloqueado hasta que se pulse un botón del ratón sobre el icono.*

procedure Apagar.Icono;

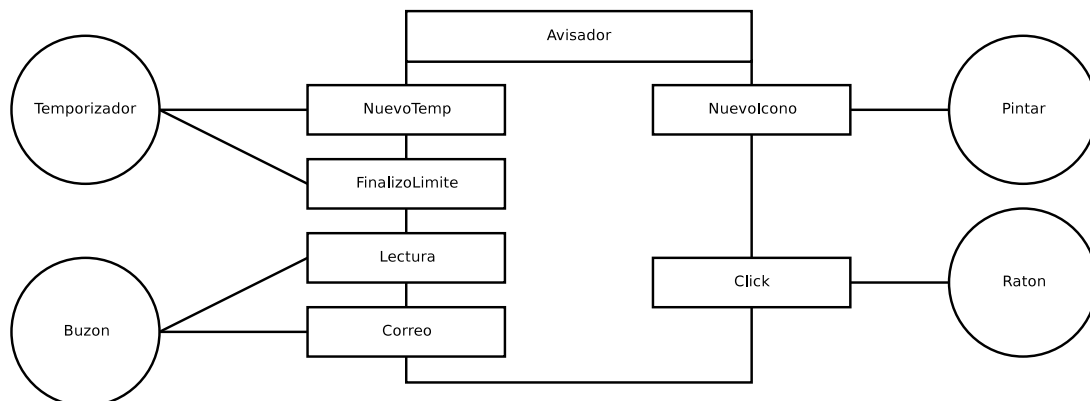
procedure Iluminar.Icono;

procedure Intermitente.Icono;

— *Hacen que el icono del correo se apague, se ilumine, o parpadee intermitentemente.*

— *Regresan inmediatamente.*

A continuación se muestra el grafo de procesos y recursos:



El código las tareas identificadas (donde R se refiere al recurso compartido):

<pre> — Buzón Tam := Tam_Correo; if Tam > 0 then Hora(H); Correo(R, H + L); end if; loop Tam1 := Cambio_Correo; if Tam1 > Tam then Hora(H); Correo(R, H + L); end if; if Tam1 = 0 then Lectura(R); end if; Tam := Tam1; end loop; </pre>	<pre> — Pintar Estado := apagado; loop case Estado of apagado: Apagar_Icono; iluminado: Iluminar_Icono; interm: Intermitente_Icono; end case; NuevoIcono(R, Estado); end loop; </pre>	<pre> — Temporizador loop Nuevo_Temp(R, Limite); Esperar_Hasta(Limite); Finalizo_Limite(R); end loop; — Click loop Espera_Raton; Click(R); end loop; </pre>
--	---	--

Y la especificación formal del recurso:

C-TADSOL Avisador

OPERACIONES

ACCIÓN Click: $TipoAvisador[es]$
ACCIÓN NuevoIcono: $TipoAvisador[es] \times Tipo_Icono[es]$
ACCIÓN Correo: $TipoAvisador[es] \times Tipo_Tiempo[e]$
ACCIÓN Lectura: $TipoAvisador[es]$
ACCIÓN NuevoTemp: $TipoAvisador[es] \times Tipo_Tiempo[s]$
ACCIÓN FinalizoLimite: $TipoAvisador[es]$

SEMÁNTICA

DOMINIO:

TIPO: $TipoAvisador = (vacía: \mathbb{B} \times icono: Tipo_Icono \times$
 $limite: Tipo_Tiempo \times nuevoolimite: \mathbb{B})$
 $Tipo_Icono = apagado \mid iluminado \mid intermitente$

INVARIANTE: $\forall r \in TipoAvisador \bullet r.vacía \rightarrow r.icono = apagado$

INICIAL(r): $r = (true, apagado, -, -)$

CPRE: $e \neq r.icono$

NuevoIcono(r, e)

POST: $r^{ent} = r^{sal} \wedge r^{ent}.icono = e^{sal}$

CPRE: *Cierto*

Click(r)

POST: $r^{sal} = r^{ent} \setminus r^{sal}.icono = apagado$

CPRE: *Cierto*

Correo(r, l)

POST: $r^{sal} = r^{ent} \setminus$
 $(r^{ent}.icono = apagado \wedge r^{ent}.vacía$
 $\rightarrow \neg r^{sal}.vacía) \wedge$
 $r^{sal}.icono = iluminado \wedge$
 $r^{sal}.limite = l \wedge r^{sal}.nuevolimite$

CPRE: ...

Lectura(r)

POST: ...

CPRE: $r.nuevolimite$

NuevoTemp(r, l)

POST: $l^{sal} = r^{ent}.limite \wedge$
 $r^{sal} = r^{ent} \setminus \neg r^{sal}.nuevolimite$

CPRE: *Cierto*

FinalizoLimite(r)

POST: $r^{sal} = r^{ent} \setminus$
 $\neg r^{sal}.nuevolimite \wedge$
 $(\neg r^{ent}.vacía \wedge r^{ent}.icono = iluminado$
 $\rightarrow r^{sal}.icono = intermitente)$

Apellidos: _____

Nombre: _____ Número de matrícula: _____

2. Cuestionario

[5 puntos]

Importante: sólo hay una respuesta válida por pregunta excepto para la pregunta 2.1. Para el resto, toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada. Toda pregunta incorrectamente contestada restará del examen **0.25 puntos**.

2.1. Avisador de correo

[1 punto]

Especificar la operación Lectura del recurso compartido Avisador de forma que se complete el diseño exigido en el enunciado *Avisador de correo* del apartado 1.

CPRE:	
CPRE informal:	
POST:	
POST informal:	

2.2. Comunicación tras *Rendez-vous*

[1 punto]

Según la metodología estudiada en clase para implementar recursos compartidos mediante el uso de *rendez-vous* y paso de mensajes síncrono, en general el cliente realiza una llamada a una entrada de la **select** del servidor para iniciar la operación. En esta llamada se pasan como parámetros los argumentos de la operación del recurso compartido más un canal que el cliente ha generado y por el que espera la respuesta del servidor. Se ha invertido este esquema siendo el servidor quien genera el canal y lo devuelve como dato de salida del **accept**:

```
— Código en el cliente para Op
— X datos de entrada de Op
— Y datos de salida de Op
declare
  CC : Channel;
begin
  Servidor.Op (X, CC);
  Receive (CC, Y);
  Destroy (CC);
end;
```

```
— Entrada en el servidor para Op
when True =>
  accept Op (X : in Xs,
            CS : out Channel) is
    Create (CS);
    — Almacenar X y CS en un contenedor
  end Op;
```

```
— Porción de código para el desbloqueo de
— Op en el servidor tras extraer X y CS
— del contenedor y comprobar la
— CPRE de la operación Op para X
<< Código para alcanzar la POST de Op >>;
Send (CS,Y);
```

Se pide: marcar cuál de las siguientes afirmaciones es verdadera.

- | | |
|--|--|
| <input type="checkbox"/> Si el cliente y el servidor se ejecutan en un entorno distribuido donde no hay compartición de memoria, el cliente no puede liberar el canal, que ha sido creado por el servidor. | <input type="checkbox"/> El accept no permite datos de salida. |
| | <input type="checkbox"/> El nombre del canal debe ser idéntico en el cliente y en el servidor. |
| | <input type="checkbox"/> Un esquema así no compilaría en Ada 95. |

2.3. Indexación de familia de *entries*

[1 punto]

Supongamos un sistema concurrente implementado en Ada 95 siguiendo la metodología explicada en clase. En dicho sistema hay un recurso compartido R con una operación $Op(R, x)$ cuya CPRE es $P(x)$. x sólo puede tomar M posibles valores. En dicho sistema hay N procesos que podrían invocar

simultáneamente la operación *Op* del recurso. Se ha optado por la siguiente implementación utilizando objetos protegidos:

```

type Xs is range 1 .. M;
entry Op (X : Xs) when True is
begin
  requeue Op_Aplazada (X);
end Op;

entry Op_Aplazada (for I in Xs) (X : Xs)
when Comprueba_P (I) is
begin
  — Código para alcanzar la POST
  ...
end Op;

```

Se pide: marcar cuál de las siguientes afirmaciones es verdadera.

- | | |
|---|---|
| <input type="checkbox"/> La guarda debería ser Comprueba.P(X). | <input type="checkbox"/> Los procesos son atendidos en estricto orden de llegada. |
| <input type="checkbox"/> No compila porque sobra la declaración de argumentos (X : Xs) de la <i>entry</i> Op_Aplazada | <input type="checkbox"/> Ninguna de las anteriores. |

2.4. Desbloques

[1 punto]

El recurso del multibuffer admite insertar o extraer un número variable de datos de un buffer acotado. Las condiciones de sincronización de ambas operaciones son que haya suficientes huecos para almacenar los datos que se quiere insertar y que haya suficientes elementos para retirar los datos que se quiere extraer. Se ha implementado el recurso mediante *rendez-vous* y paso de mensajes siguiendo quedando el siguiente código de desbloqueo tras la *select*:

```

...
end select;
<< Bucle de desbloqueo de todos los procesos bloqueados para extraer >>;
<< Bucle de desbloqueo de todos los procesos bloqueados para insertar >>;
end loop;

```

Se pide: marcar cuál de las siguientes afirmaciones es verdadera.

- | | |
|--|---|
| <input type="checkbox"/> El esquema puede provocar que queden procesos bloqueados a pesar de que se cumple la su CPRE. | <input type="checkbox"/> El esquema de código es perfecto. |
| <input type="checkbox"/> Es necesario desbloquear primero a los procesos bloqueados para insertar. | <input type="checkbox"/> Hay que desbloquear alternativamente a procesos de diferente tipo: consumidor, productor, consumidor, productor, ... |

2.5. ¿Sincronía?

[1 punto]

Según G. R. Andrews y F. B. Schneider, “[...] si un canal de comunicación tiene una capacidad de almacenamiento ilimitada, en la práctica permite que los procesos que ejecutan un *send* nunca queden bloqueados y a este tipo de paso de mensajes se le denomina asíncrono. [...] En el otro extremo, cuando el proceso que invoca un *send* queda bloqueado hasta que el correspondiente *receive* sea ejecutado, se habla de paso de mensajes síncrono.”

Dada la siguiente implementación de canales

```

protected type Channel is
  entry Send (M : in Message);
  entry Receive (M : out Message);
private
  Empty : Boolean := True;
  Data : Message;
end Channel;

protected body Channel is
  entry Send (M : in Message) when Empty is
  begin
    Data := M; Empty := False;
  end Send;
  entry Receive (M : out Message) when not Empty is
  begin
    M := Data; Empty := True;
  end Receive;
end Channel;

```

Se pide: marcar cuál de las siguientes afirmaciones es verdadera según la definición anterior.

- | | |
|--|---|
| <input type="checkbox"/> Implementa paso de mensajes síncrono. | <input type="checkbox"/> Implementa paso de mensajes síncrono y paso de mensajes asíncrono. |
| <input type="checkbox"/> No implementa ni paso de mensajes síncrono ni paso de mensajes asíncrono. | <input type="checkbox"/> Implementa paso de mensajes asíncrono. |

Este examen consta de tres apartados: un planteamiento de un problema con una solución incompleta, un cuestionario y una pregunta de implementación. La duración total es de **tres horas**. El cuestionario debe contestarse **en la misma hoja** en que se os entrega (consultad con un profesor si necesitáis otra hoja) y se podrá devolver hasta **una hora** después de recibirlo. Al hacerlo se os dará la pregunta de implementación.

La puntuación total del examen es de **10 puntos**. Las soluciones se proporcionarán antes de la revisión. Las calificaciones se darán a conocer el día **20 de junio**. La revisión del examen tendrá lugar el día **22 de junio**.

3. Implementación

[5 puntos]

Para cada uno de los apartados siguientes ha de entregarse lo que se pide en ellos. Recordad **entregar la respuesta a cada apartado en un juego de hojas separadas**.

3.1. Objetos protegidos

[2.5 puntos]

Realizar la implementación de las operaciones `Nuevo.Icono`, `Correo` y `Finalizo.Limite` del recurso compartido `Avisador` utilizando objetos protegidos como mecanismo de concurrencia, aplicando la metodología propia de la asignatura y siguiendo el esquema que se ofrece en el apartado 3.3.

3.2. *Rendez-Vous* / Paso de mensajes

[2.5 puntos]

Completar toda la implementación de las operaciones `Nuevo.Icono`, `Correo` y `Finalizo.Limite` del recurso compartido `Avisador` utilizando *rendez-vous* y paso de mensajes como mecanismos de concurrencia, aplicando la metodología propia de la asignatura y siguiendo el esquema que se ofrece en el apartado 3.3.

3.3. Esquema para completar la implementación

3.3.1. `avisador.ads` (se adjunta para más información; no hay que modificarlo o ampliarlo, sólo respetarlo)

```
with Tiempo;  
use Tiempo;  
  
package Avisador is  
  type Tipo_Icono is (Apagado, Iluminado, Intermitente);  
  type Tipo_Avisador is private;  
  
  procedure Nuevo_Icono (R : in out Tipo_Avisador; E : in out Tipo_Icono);  
  procedure Correo (R : in out Tipo_Avisador; L : in Tipo_Tiempo);  
  procedure Finalizo_limite (R : in out Tipo_Avisador);  
  
private  
  type Avisador_Impl;  
  type Tipo_Avisador is access Avisador_Impl;  
end Avisador;
```

3.3.2. avisador.adb (esquema)

— *Importación de paquetes que sean necesarios: a completar*

package body Avisador **is**

— *Tipos auxiliares e instanciación de paquetes genéricos: a completar*

— *Declaración del tipo como objeto protegido o como servidor*
protected ó **task type** Avisador_Impl **is**
 — *A completar*
end Avisador_Impl;

— *Implementación del tipo como objeto protegido o como servidor*
protected ó **task body** Avisador_Impl **is**
 — *A completar*
end Avisador_Impl;

function Crear_Avisador **return** Tipo_Avisador **is**
 — *A completar*
begin
 — *A completar*
end Crear_Avisador;

procedure Nuevo_Icono (R : **in out** Tipo_Avisador; E : **in out** Tipo_Icono)
 — *A completar*
begin
 — *A completar*
end Nuevo_Icono;

procedure Click (R : **in out** Tipo_Avisador) **is**
 — *A completar*
begin
 — *A completar*
end Click;

procedure Correo (R : **in out** Tipo_Avisador; L : **in** Tipo_Tiempo) **is**
 — *A completar*
begin
 — *A completar*
end Correo;

procedure Lectura (R : **in out** Tipo_Avisador)
 — *A completar*
begin
 — *A completar*
end Lectura;

procedure Nuevo_Temp (R : **in out** Tipo_Avisador; L : **out** Tipo_Tiempo) **is**
 — *A completar*
begin
 — *A completar*
end Nuevo_Temp;

procedure Finalizo_limite (R : **in out** Tipo_Avisador)
 — *A completar*
begin
 — *A completar*
end Finalizo_Limite;
end Avisador;