Examen de Programación Concurrente - Clave a

Febrero 2008

Departamento de Lenguajes, Sistemas Informáticos e Ingeniería del *Software*

Normas

Este examen es un cuestionario tipo test que consta de **10 preguntas** en **4 páginas**. La puntuación total del examen es de **10 puntos**. La duración total es de **una hora y media**. El examen debe contestarse en las **hojas de respuestas**. No olvidéis rellenar **apellidos, nombre y DNI** en cada hoja de respuesta.

Sólo hay una respuesta válida por pregunta. Toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada. Toda pregunta incorrectamente contestada restará del examen una cantidad de puntos igual a la puntuación de la pregunta dividido por el número de alternativas ofrecidas en la misma.

La solución al examen se proporcionará antes de la revisión. Las calificaciones se darán a conocer el **19 de febrero**. La revisión del examen tendrá lugar el **21 de febrero**.

Cuestionario

(1 punto) 1. Dado el siguiente programa concurrente:

```
procedure Examen is

   X : Integer := 2;
   Y : Integer := 3;
   M : Bin_Semaphore;

   task A;
   task B;

   task body A is
   begin
      Wait (M);
      X := X + Y;
      Signal (M);
   end A;

   task body B is
   begin
      Wait (M);
      Y := X * Y;
      Signal (M);
   end B;


begin  -- Programa principal
   null;
end Examen;
```

Se pide marcar la afirmación correcta sobre el valor del par $(\texttt{X}, \texttt{Y})$ tras la terminación de las tareas `A` y `B`.

(a) $(5, 15)$ no es un valor posible.

(b) $(8, 6)$ no es un valor posible.

(c) Cualquier valor de las variables es posible por ser un programa concurrente.

(d) $(15, 5)$ no es un valor posible.

(1 punto) 2. Supongamos que en el código de la pregunta 1 eliminamos el semáforo binario y todo su código asociado. Consideremos que cada sentencia de asignación se realiza en tres pasos atómicos: lectura de variables a la derecha, suma o multiplicación de los valores leidos y escritura del resultado en la variable a la izquierda.

Se pide marcar la afirmación correcta sobre el valor del par $(\texttt{X}, \texttt{Y})$ tras la terminación de las tareas `A` y `B`.

(a) $(5, 6)$, $(5, 15)$ y $(8, 6)$ son todos los posibles valores.

(b) $(8, 6)$ no es un valor posible.

(c) Cualquier valor de las variables es posible por ser un programa concurrente.

(d) $(5, 15)$ no es un valor posible.

(1 punto) 3. Dado el siguiente **CTAD** (no se muestra el interfaz pero no contiene otras operaciones que las especificadas, el tercer componente del tipo es una tabla):

TIPO: $Barrera = (abierta : \mathbb{B} \times color\_abierto : Color \times quieren\_pasar : Color \nrightarrow \mathbb{Z})$

DONDE: $Color = \{rojo, verde, azul\}$

INVARIANTE: $\forall b \in Barrera.\ \forall c \in Color.\ 0 \leq b.quieren\_pasar(c) \wedge b.quieren\_pasar(c) \leq 10$

INICIAL(b): $\neg b.abierta \wedge \forall c \in Color.\ b.quieren\_pasar(c) = 0$
CPRE: $\neg b.abierta$
SolicitarPaso(b,c)
POST: $b^{sal}.abierta = b^{ent}.abierta \wedge b^{sal}.color\_abierto = b^{ent}.color\_abierto$
$\wedge\ b^{sal}.quieren\_pasar = b^{ent}.quieren\_pasar \oplus \{c \mapsto b^{ent}.quieren\_pasar(c) + 1\}$
CPRE: $b.abierta \wedge b.color\_abierto = c$
Pasar(b,c)
POST: $b^{sal}.abierta = (b^{sal}.quieren\_pasar(c) > 0) \wedge b^{sal}.color\_abierto = b^{ent}.color\_abierto$
$\wedge\ b^{sal}.quieren\_pasar = b^{ent}.quieren\_pasar \oplus \{c \mapsto b^{ent}.quieren\_pasar(c) - 1\}$
CPRE: $\neg b.abierta$
Abrir(b,c)
POST: $b^{sal}.abierta = (b^{ent}.quieren\_pasar(c) > 0) \wedge b^{sal}.color\_abierto = c$
$\wedge\ b^{sal}.quieren\_pasar = b^{ent}.quieren\_pasar$

Supóngase un programa concurrente en el que los procesos respetan los siguentes protocolos (o esquemas de llamada): $SolicitarPaso(b, c_1)$; $Pasar(b, c_1)$ y $Abrir(b, c_2)$ donde c_1 y c_2 son colores.

Se pide señalar la respuesta correcta (asumir, obviamente, que el invariante se cumple antes de la invocación de cada operación).

(a) El programa puede violar el invariante del recurso compartido sólo en la operación $SolicitarPaso$.

(b) El programa puede violar el invariante del recurso compartido sólo en la operación $Pasar$.

(c) El programa puede violar el invariante del recurso compartido sólo en la operación $Abrir$.

(d) Ninguna de las otras respuestas es correcta.

(1 punto) 4. En el programa concurrente de la pregunta 3 se pretende evitar la violación del invariante del recurso.

Se pide señalar la mejor decisión.

(a) Poner $\neg b.abierta \wedge b.quieren\_pasar(c) < 10$ como CPRE de $SolicitarPaso$.

(b) Poner $b.abierta \wedge b.quieren\_pasar(c) > 0$ como CPRE de $Pasar$.

(c) Eliminar el invariante.

(d) Poner $\neg b.abierta \wedge b.quieren\_pasar(c) \leq 10$ como CPRE de $SolicitarPaso$.

(1 punto) 5. 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 TChannel is
   entry Send (M : in Message);
   entry Receive (M : out Message);
private
   Empty : Boolean := True;
   Data : Message;
end TChannel;

protected body TChannel 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;
```

Se pide marcar la afirmación correcta sobre la implementación mostrada.

(a) Implementa paso de mensajes síncrono.

(b) No implementa ni paso de mensajes síncrono ni paso de mensajes asíncrono.

(c) Implementa paso de mensajes síncrono y paso de mensajes asíncrono.

(d) Implementa paso de mensajes asíncrono.

(1 punto) 6. Dado el tipo objeto protegido del problema 5 y el siguiente programa concurrente:

```
C : TChannel;

task type T;

task body T is
   M : Message := 0;
begin
   loop
      C.Send (M);
   end loop;
end T;

T1, T2 : T;

begin  --  Programa principal
   null;
end Examen;
```

Se pide señalar la afirmación correcta.

(a) La entrada `Receive` del objeto protegido `C` nunca llega a ser ejecutada.

(b) La entrada `Receive` del objeto protegido `C` se ejecutará nada más terminar la ejecución de la entrada `Send`.

(c) Sólo uno de los procesos se queda bloqueado en la entrada `Send` de `C`.

(d) Ningún proceso queda bloqueado.

(1 punto) 7. La condición de sincronización de una operación de un recurso compartido depende de un dato de entrada en un intervalo de datos entre 1 y `N`. Dicha operación va a ser invocada sólo por un proceso. **Se pide** señalar la afirmación correcta.

(a) La implementación de la operación mencionada mediante objetos protegidos en Ada exige introducir una familia de entradas indexada con el tipo `1 .. N`

(b) La implementación de la operación mediante objetos protegidos en Ada es viable introduciendo una única entrada aplazada (al margen de cómo se implementen el resto de las operaciones).

(c) Si `N` fuera un número muy grande, la implementación de dicha operación mediante objetos protegidos en Ada exigiría introducir una familia de entradas indexada por el número de procesos (aunque este número sea 1).

(d) Ninguna de las otras respuestas es correcta puesto que no nos encontraríamos ante un programa concurrente.

(1 punto) 8. El siguiente código es, supuestamente, fruto de la aplicación de la metodología enseñada en clase para implementar el recurso del problema 3 utilizando *rendez-vous* y paso de mensajes síncrono:

```
            or when True =>
               accept Pasar (Col : in Color; Ch : in out Channel) do
                  Insertar (Esperan_Pasar, (Col, Ch));
               end Pasar;
         end select;
         while not Es_Vacia (Esperan_Pasar) loop
            Primero (Esperan_Pasar, D);
            Borrar (Esperan_Pasar);
            if Abierta and Color_Abierto = D.Col then
               Quieren_Pasar (D.Col) := Quieren_Pasar (D.Col) - 1;
               Abierta := Quieren_Pasar (D.Col) > 0;
               Send (D.Ch, True);
            else
               Insertar (Esperan_Pasar, D);
            end if;
         end loop;
      end loop;
```

Se pide señalar la afirmación correcta.

(a) Es un ejemplo de perfecta aplicación de la metodología.

(b) Es necesario enviar el color en el mensaje.

(c) El servidor puede dejar de atender peticiones.

(d) Para este caso el bucle de desbloqueo va a terminar después de despertar a todos los procesos que pueden ser desbloqueados tal y como exige la metodología explicada en la asignatura.

(1 punto) 9. El siguiente código es, supuestamente, fruto de la aplicación de la metodología enseñada en clase para implementar el recurso del problema 3 utilizando *rendez-vous* y paso de mensajes síncrono. Cabe resaltar que en lugar de una única colección de pares color-canal se utiliza una colección de canales por cada color (en forma de array de colas) para que el servidor recuerde las peticiones de ejecución de la operación `Pasar`.

```
            or when True =>
               accept Pasar (Col : in Color; Ch : in out Channel) do
                  Insertar (Esperan_Pasar (Col), Ch);
               end Pasar;
         end select;
         if Abierta and not Es_Vacia (Esperan_Pasar (Color_Abierto)) then
            while not Es_Vacia (Esperan_Pasar (Color_Abierto)) loop
               Primero (Esperan_Pasar (Color_Abierto), Ch);
               Borrar (Esperan_Pasar (Color_Abierto));
               Quieren_Pasar (Color_Abierto) :=
                 Quieren_Pasar (Color_Abierto) - 1;
               Send (Ch, True);
            end loop;
            Abierta := Quieren_Pasar (Color_Abierto) > 0;
         end if;
      end loop;
```

Se pide señalar la afirmación correcta.

(a) En Ada 95 no es posible tener arrays de colas.

(b) Es necesario enviar el color en el mensaje.

(c) El servidor puede dejar de atender peticiones.

(d) Para este caso el bucle de desbloqueo va a terminar después de despertar a todos los procesos que pueden ser desbloqueados tal y como exige la metodología explicada en la asignatura.

Nota: esta pregunta fue anulada en el examen porque el código contenía un error que hacía que todas las respuestas fueran no válidas. Aquí se presenta una versión corregida.

(1 punto) 10. Se pretende que una tarea que ejecute el código **`loop`** `...; Esperar(R); ...;` **`end loop;`** se detenga en la llamada a la operación `Esperar` hasta que se cumpla una condición `Cond`. La llamada se ha descompuesto en el cliente (procedimiento `Esperar`) e implementado en el servidor (tarea `R`) utilizando *rendez-vous* y paso de mensajes síncrono de la siguiente forma:

```
procedure Esperar (R : Recurso);
   C : Channel;
begin
   Create (C);
   R.Esperar (C);
   Send (C, False);
   Destroy (C);
end Esperar;
```

```
--  Código del servidor R
select
   when Cond =>
      accept Esperar (C: in out Channel) do
         Receive (C, V);
      end Esperar;
or
   ...
end select;
```

Se pide marcar cuál de las siguientes afirmaciones es la correcta.

(a) Los procesos se bloquean porque el cliente debe hacer un `Receive` y el servidor un `Send`. Es decir, deben intercambiarse las llamadas que hacen ahora.

(b) Funciona, sólo se bloquea el cliente cuando la condición `Cond` no se cumple.

(c) El código produce siempre un interbloqueo del cliente y el servidor.

(d) Se bloquea, pero funcionaría si el cliente envía un `True` en lugar de un `False` para hacer cierta la condición `Cond`.