Examen de Programación Concurrente - Clave: a

Junio 2008

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

Normas

Este examen es un cuestionario tipo test que consta de **10 preguntas** en **6 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, DNI y clave del examen**.

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 **2 de julio**. La revisión del examen tendrá lugar el **4 de julio**.

Cuestionario

(1 punto) 1. La siguiente función va a ser ejecutada simultaneamente por varios procesos.

```
function MCD (X, Y : Natural) return Natural is
   A : Natural := X;
   B : Natural := Y;
begin
   while B > 0 loop
      if A > B then
         A := A - B;
      else
         B := B - A;
      end if;
   end loop;
   return A;
end MCD;
```

Se pide marcar la afirmación correcta.

(a) Es necesario asegurar exclusión mutua en el acceso a las variables `A` y `B`.

(b) No es necesario asegurar exclusión mutua en el acceso a las variables `A` y `B`.

(1 punto) 2. Dado el siguiente programa concurrente:

```
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;
```

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) $(5, 6)$ no es un valor posible.

(c) Ninguna de las otras respuestas es correcta.

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

(1 punto) 3. Supóngase que se quiere implementar un programa que lea un número natural menor que 1000 y cree exactamente tantas tareas como dicho número indique.

Se pide marcar la afirmación correcta.

(a) No es posible realizar tal programa en Ada puesto que la creación de tareas es estática y debe ser decidida en tiempo de compilación.

(b) No es posible realizar tal programa en Ada porque hacen falta punteros a tareas y Ada no tiene.

(c) Es posible hacerlo en Ada usando punteros a tareas.

(d) Es posible hacerlo en Ada pero la única forma de hacerlo es creando un array de 1000 tareas.

(1 punto) 4. La instrucción TST (*Test and set*) es típica de algunas arquitecturas. Su comportamiento viene dado por la existencia de una variable global `C`. Al ejecutar `TST(L)`, donde `L` es una variable, tiene lugar de forma atómica la siguiente ejecución: `L := C; C := 1`. El siguiente programa concurrente hace uso de dicha instrucción para implementar acceso a una sección crítica:

```
procedure Test_And_Set is
   C : Integer := 0;
   procedure TST (L : out Integer);

   task T1;
   task T2;

   task body T1 is
      L : Integer;
   begin
      loop
         S1;
         loop
            TST (L);
            exit when L = 0;
         end loop;
         Sec_Critica;
         C := 0;
      end loop;
   end T1;

   task body T2 is
      L : Integer;
   begin
      loop
         S2;
         loop
            TST (L);
            exit when L = 0;
         end loop;
         Sec_Critica;
         C := 0;
      end loop;
   end T2;

begin
   null;
end Test_And_Set;
```

Se pide marcar la afirmación correcta.

(a) No se garantiza la propiedad de exclusión mutua.

(b) Se puede producir interbloqueo.

(c) Se puede producir inanición.

(d) Ninguna de las otras respuestas es correcta.

(1 punto) 5. A continuación se muestra una implementación de lectores/escritores realizada con semáforos (`L` indica el número de lectores leyendo, `E` indica que hay un escritor escribiendo y `Mutex` protege el acceso a `L`).

```
procedure Lectores_Escritores is

   L : Integer := 0;
   E : Bin_Semaphore;
   Mutex : Bin_Semaphore;

   task type Escritor;
   task type Lector;

   E1, E2 : Escritor;
   L1, L2, L3, L4 : Lector;

   task body Escritor is
   begin
      loop
         Wait (E);
         Escribir_Datos;
         Signal (E);
      end loop;
   end Escritor;

   task body Lector is
   begin
      loop
         Wait (Mutex);
         L := L + 1;
         if L = 1 then
            Wait (E);
         end if;
         Signal (Mutex);
         Leer_Datos;
         Wait (Mutex);
         L := L - 1;
         if L = 0 then
            Signal (E);
         end if;
         Signal (Mutex);
      end loop;
   end Lector;

begin
   null;
end Lectores_Escritores;
```

Se pide marcar la afirmación correcta.

(a) No se cumple la propiedad de seguridad (propiedad que dice que dos procesos nunca ejecutan simultaneamente `Escribir_Datos` ni `Escribir_Datos` y `Leer_Datos`).

(b) Se puede producir interbloqueo.

(c) Se puede producir inanición.

(d) No se garantiza la propiedad de exclusión mutua en el acceso a la variable `L`.

(1 punto) 6. Supóngase para el código de la pregunta 5 una implementación *FIFO* de los semáforos binarios (el primero en quedar bloqueado en un **`Wait`** es el primero en ser desbloqueado tras un **`Signal`**).

Se pide marcar la afirmación correcta.

(a) No se cumple la propiedad de seguridad (propiedad que dice que dos procesos nunca ejecutan simultaneamente `Escribir_Datos` ni `Escribir_Datos` y `Leer_Datos`).

(b) Se puede producir interbloqueo.

(c) Se puede producir inanición.

(d) No se garantiza la propiedad de exclusión mutua en el acceso a la variable `L`.

(1 punto) 7. Suponiendo que estamos usando objetos protegidos (O.P.) para implementar recursos compartidos, señalar cuál de las siguientes afirmaciones es correcta:

(a) El recurso se debe encapsular siempre dentro de un O.P.

(b) Si el recurso se encapsula dentro de un O.P., sólo lo puede usar un proceso a la vez.

(c) Aunque se encapsule el recurso dentro de un O.P., varios procesos pueden usarlo a la vez.

(d) Siempre debe definirse fuera del O.P. y usar el O.P. para gestionar los permisos de acceso.

(1 punto) 8. Dado el siguiente tipo de objeto protegido

```
protected type Complex is
   procedure Set_Value (Re : Float;
                        Im : Float);
   function Get_Re return Float;
   function Get_Im return Float;
   function Get_Ab return Float;
private
   Real, Imaginary : Float := 0.0;
end Complex;

protected body Complex is
   procedure Set_Value (Re : Float;
                        Im : Float) is
   begin
      Real := Re; Imaginary := Im;
   end Set_Value;
   function Get_Re return Float is
   begin
      return Real;
   end Get_Re;
   function Get_Im return Float is
   begin
      return Imaginary;
   end Get_Im;
   function Get_Ab return Float is
   begin
      return Sqrt (
         Real * Real
         + Imaginary * Imaginary);
   end Get_Ab;
end Complex;
```

Supóngase la existencia de dos tareas `T11` y `T12` trabajando con un objeto protegido `C1` y una tarea `T2` trabajando con un objeto protegido `C2` (`C1` y `C2` son de tipo `Complex`).

Se pide marcar la afirmación correcta.

(a) Cuando `T11` ejecuta `C1.Set_Value (...)` una llamada a `C2.Set_Value (...)` en `T2` queda bloqueada hasta que `T11` termine la ejecución de `C1.Set_Value (...)`.

(b) Cuando `T11` ejecuta `C1.Set_Value (...)` una llamada a `C2.Get_Ab (...)` en `T2` queda bloqueada hasta que `T11` termine la ejecución de `C1.Set_Value (...)`.

(c) Cuando `T11` ejecuta `C1.Set_Value (...)` una llamada a `C1.Get_Ab` en `T12` queda bloqueada hasta que `T11` termine la ejecución de `C1.Set_Value (...)`.

(d) Ninguna de las otras respuestas es correcta.

(1 punto) 9. El siguiente esquema de código con dos entradas de un objeto protegido en Ada implementa una operación de un recurso compartido cuya precondición de concurrencia depende de su parámetro de entrada:

```
protected R is
   --  Operaciones públicas
   entry Op (X : T);
   [...];
private
   -- Operaciones privadas
   entry Op_Aplazada (X : T);
   -- Estado
   [...];
   Op_Cerrada : Boolean := False;
   Copia_De_X : T;
end R;
```

```
protected body R is
   [...]
   entry Op (X : T)
   when not Op_Cerrada is
   begin
      Op_Cerrada := True;
      Copia_De_X := X;
      requeue Op_Aplazada;
   end Op;

   entry Op_Aplazada (X : T)
   when P(Copia_De_X) is
   begin -- Implementación de Op
      [Bloque Op];
      Op_Cerrada := False;
   end Op_Aplazada;
end R;
```

Supongamos que $P(X)$ es una expresión Ada que comprueba la precondición de concurrencia de `Op` para un dato de entrada X, que $[Bloque\ Op]$ es una implementación correcta de dicha operación y que `R` es una variable del tipo del objeto protegido.

Se pide marcar la afirmación correcta.

(a) Los procesos que ejecuten `R.Op(`E`)` pueden quedar bloqueados en dicha entrada aunque la expresión $P(E)$ se evalúe a `True`.

(b) Es un esquema válido sea cual séa el conjunto de requisitos del sistema.

(c) El esquema lleva a una violación de las propiedades de seguridad del recurso compartido.

(d) El esquema es incorrecto porque Ada exige definir `Op_Aplazada` como una familia de entradas con T como tipo discreto en la definición.

(1 punto) 10. Se pretende modificar el esquema de código utilizado para implementar recursos compartidos mediante *rendez-vous* y paso de mensajes para utilizar sólo *rendez-vous*. La idea es la siguiente: por cada operación del recurso en la que fuera necesario el uso de un canal, se ha substituido dicho uso por una nueva entrada en el servidor. Dicha entrada es invocada por el cliente tras invocar la entrada principal asociada a dicha operación. El servidor no aceptará la entrada aplazada hasta el momento en el que sea posible. Veamos el esquema de código:

```
procedure Op (R : in out Recurso;
              X : in     T;
              Y :    out T) is
begin
   R.S.Op (X);
   R.S.Op_Aplazada (Y);
end Op;

task body Servidor is
   --  ...

   X : T := 0;
   R : T;
begin
   loop
      select
         when True =>
            accept Op (X : T) do
               Almacenar (X);
            end Op;
      or --  ...

      end select;
      --  Atención de tareas bloqueadas
      while No_Mas_Por_Atender loop
         Recuperar (X);
         if CPre_De_Op (X) then
            Realizar_Op (X, R);
            Borrar (X);
            accept Op_Aplazada
                      (Y : out T) do
               Y := R;
            end Op_Aplazada;
         end if;
      end loop;
   end loop;
end Servidor;
```

La aplicación del resto de la metodología aprendida en el curso es correcta (el bucle de desbloqueo, los parámetros de entrada salida, la comprobación de la precondición de concurrencia, la realización de la operación en si, el almacenamiento y recuperación de los datos, etc.).

Se pide marcar la afirmación correcta.

(a) La linea **accept** `Op_Aplazada` en el servidor es un error de sintaxis en Ada 95.

(b) El funcionamiento del programa es el mismo que con el canal.

(c) Puede desbloquearse un proceso para el que no se cumple la precondición de concurrencia.

(d) El esquema presentado provoca interbloqueo.