

Examen de Programación Concurrente - Clave a
Septiembre 2007
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**. El examen debe contestarse en las **hojas de respuestas**. No olvidar 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 **18 de septiembre**. La revisión del examen tendrá lugar el **20 de septiembre**.

Cuestionario

- (1 punto) 1. Con respecto al análisis de propiedades de seguridad y vivacidad de programas concurrentes. ¿Supone alguna diferencia que los procesos del programa se ejecuten sobre diferentes procesadores en paralelo o en un único procesador mediante multitarea? **Se pide** señalar la respuesta correcta.

- (a) Sí.
- (b) No.

- (1 punto) 2. Tenemos el siguiente código que será ejecutado por dos tareas concurrentes. Supondremos que las asignaciones son atómicas y que X e Y son variables compartidas entre dichos procesos:

```
procedure Swap is
  Z : Integer;
begin
  Z := X; X := Y; Y := Z;
end Swap;
```

Imaginemos que en el momento en que las dos tareas llaman a `Swap` los valores de las variables compartidas son $X = 1$, $Y = 2$. ¿Qué posibles valores pueden tener X e Y al final de la ejecución de `Swap`? **Se pide** señalar la respuesta correcta.

- (a) $X=1, Y=2$ o $X=2, Y=1$ o $X=1, Y=1$ o $X=2, Y=2$
- (b) $X=1, Y=2$ o $X=2, Y=1$ (y ningún otro)
- (c) $X=1, Y=1$ o $X=2, Y=2$ (y ningún otro)
- (d) $X=2, Y=1$ o $X=1, Y=1$ o $X=2, Y=2$ (y ningún otro)

- (1 punto) 3. Supóngase un programa concurrente con un objeto protegido R del tipo T

```
protected type T is
  entry A;
  entry B;
private
  As : Natural := 0; Bs : Natural := 0;
end T;
```

```

protected body T is
  entry A when True is
  begin
    As := As + 1; As := As - 1;
  end A;
  entry B when True is
  begin
    Bs := Bs + 1; Bs := Bs - 1;
  end B;
end T;

```

dos procesos ejecutando la entrada R.A en sendos bucles infinitos y dos procesos ejecutando la entrada R.B en sendos bucles infinitos.

Se pide señalar la afirmación incorrecta.

- (a) Alguna vez $R.As + R.Bs = 2$.
- (b) Siempre $R.As + R.Bs \leq 1$.
- (c) Siempre se cumple que $R.As \leq 1$.

(1 punto) 4. En un sistema de control de un edificio de N pisos se dispone de la siguiente operación:

```

-- La ejecución de la operación queda bloqueada hasta que la
-- temperatura del piso P se encuentre fuera del rango establecido
-- por Min y Max y devuelve la temperatura exacta en T
procedure Detectar_Fuera_De_Rango (P : in      Piso;
                                   Min : in      Temperatura;
                                   Max : in      Temperatura;
                                   T   : out      Temperatura);

```

¿Sería posible hacer un programa con un único proceso capaz de reaccionar en el momento en el que en algún piso del edificio la temperatura se encuentre fuera de rango?

Se pide señalar la respuesta correcta.

- (a) No.
- (b) Sí, el proceso ejecutaría la operación por cada piso en un bucle ignorando el valor de T.
- (c) Sí, el proceso ejecutaría la operación por cada piso en un bucle comprobando tras cada ejecución que T está entre Min y Max.

(1 punto) 5. **Se pide** señalar cuál de las siguientes afirmaciones es correcta:

- (a) Un objeto protegido garantiza que nunca hay dos procesos o más accediendo simultáneamente a sus datos.
- (b) Un objeto protegido garantiza que nunca hay dos procesos o más accediendo simultáneamente a sus datos desde una operación determinada pero permite el acceso simultáneo desde diferentes operaciones.
- (c) Un recurso compartido garantiza que nunca hay dos procesos o más accediendo simultáneamente a sus datos desde una operación determinada pero permite el acceso simultáneo desde diferentes operaciones.
- (d) Un recurso compartido garantiza que nunca hay dos procesos o más accediendo simultáneamente a sus datos.

- (1 punto) 6. La precondition 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 respuesta correcta.
- (a) La implementación de la operación mencionada mediante objetos protegidos en Ada exige introducir una familia de entradas indexada con el tipo $1 \dots N$
 - (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) 7. Dado el programa concurrente que forman los siguientes procesos:

```
task body T1 is
begin
  select
    accept E do
      S1;
    end E;
    S2;
  end select;
end T1;

task body T2 is
begin
  S3;
  T1.E;
  S4;
end T2;
```

Se pide señalar cuáles de las siguientes sentencias pueden ser ejecutadas simultáneamente.

- (a) S1 y S3.
 - (b) S1 y S4.
 - (c) S2 y S3.
 - (d) S2 y S4.
- (1 punto) 8. Al implementar un recurso compartido mediante *rendezvous* y paso de mensajes, el estado del recurso se representa mediante variables locales de un proceso que hace de servidor de las operaciones del recurso. **Se pide** señalar cual de las siguientes afirmaciones es la correcta:
- (a) La exclusión mutua en el acceso al estado del recurso queda automáticamente garantizada.
 - (b) Para garantizar la exclusión mutua, los procesos deben sincronizarse y no realizar simultáneamente una llamada al servidor sobre distintas entradas.
 - (c) Para garantizar la exclusión mutua, los procesos deben sincronizarse para no realizar simultáneamente una llamada al servidor sobre la misma entrada.
 - (d) El concepto de exclusión mutua desaparece al no compartir los procesos recurso alguno.
- (1 punto) 9. 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.

- (a) El esquema puede provocar que queden procesos bloqueados a pesar de que se cumple la CPRE de la operación que invocaron.
- (b) Es necesario desbloquear primero a los procesos bloqueados para insertar.
- (c) El esquema de código es perfecto.
- (d) Hay que desbloquear alternativamente a procesos de diferente tipo: consumidor, productor, consumidor, productor, ...

- (1 punto) 10. Las implementaciones basadas en objetos protegidos necesitan, en general, almacenar las peticiones que dependen de parámetros de entrada en una tabla cuya clave debe poder servir de índice de un *array*. Las basadas en paso de mensajes ofrecen mayor libertad: las peticiones pueden almacenarse en cualquier estructura de datos y los clientes se desbloquean cuando reciben un mensaje del servidor. ¿Por qué no unir ambas ideas y realizar una implementación basada en objetos protegidos, pero cuyo desbloqueo se realiza con un canal explícito? La idea se muestra en el siguiente fragmento de código, donde suponemos, como se vino haciendo durante el curso, que tanto **Send** como **Receive** trabajan sobre canales síncronos:

Recurso con objetos protegidos

```

-- CPRE: P(Estado, Arg1)
--      E(Estado, Arg1, Arg2)
-- POST: Arg2sal = Q(Estado, Arg1) ∧ ...
-- Implementación:
entry E (Arg1: in T1; Can: in out Channel)
when True is begin
  -- T es una estructura de datos cualquiera que nos venga bien
  T := Almacenar (T, (Arg1, Can));
  requeue E1;
end E;

entry E1 (....)
when PodemosServir (T) is begin
  -- Se recupera alguna petición que podamos atender
  (Arg1, Canal) := Recuperar (T);
  Send (Canal, Q (Estado, Arg1));
end E1;

```

Código del cliente

```

loop -- Cliente
  ObProt.E (MiArg1, Canal);
  Receive (Canal, MiArg2);
end loop;

```

Se pide marcar cuál de las siguientes afirmaciones sobre el esquema anterior es verdadera:

- (a) Es perfectamente válido.
- (b) No compila porque en el estado privado del objeto protegido sólo se pueden usar vectores, y no “una estructura de datos cualquiera”.
- (c) Se bloquea siempre.
- (d) Podría bloquearse en algunos casos.