Pallicalo

P. CONCURRENTE. QUIZ. 15-04-2010 **Grupo (A | B)**

Apellidos: PLANTILLA **Nombre:**

1 [1,5] Indicar (V/F):

| V | a) Con respecto a los axiomas de la operación *wait()* de los monitores: para señales con semántica desplazante no tiene porqué cumplirse el invariante del monitor como poscondición de la citada operación wait().

| F | b) Para un determinado tipo de señales se puede interrumpir la ejecución de un procedimiento de un monitor, tras la ejecución de una operación, sin que se cumpla el invariante.

| F | c) En el caso de las señales con sem ántica *no-desplazante* no se puede evitar la anomalía denominada *robo de señal*[1] por parte de los procesos que utilizan los servicios de un monitor. Sólo podemos programar los procedimientos de tal manera que el citado robo no produzca valores inconsistentes en los datos que protege el monitor.

| F | d) En ningún caso pueden estar ejecutándose concurrentemente 2 procedimientos de monitores por parte de los procesos concurrentes de un mismo programa.

| F | e) La secuencia de ejecución de un programa concurrente compuesto de un monitor y varios procesos concurrentes es equivalente a una secuencia de ejecución de los procedimientos del citado monitor en un orden aleatorio que no se puede conocer a-priori.

| F | f) Con señales SX (se fuerza la programación de los procedimientos para que los procesos dejen libre el monitor después de ejecutarse la operación *signal()*) se pueden programar soluciones a casi todos los problemas de sincronización de los programas concurrentes.

| F | g) Programando con notificaciones (≈ señales con semántica SC) la operación wait() siempre ha de programarse dentro de un while para evitar el efecto del robo de señal sobre los datos que protege el monitor.

2 [2,5] Responder (V/F) a las preguntas que se formulan a continuación con respecto al siguiente monitor que implementa una operación de comunicación por difusión (*broadcast*) para un conjunto de procesos distribuidos ejecutándose en diferentes nodos de una red local simulada.

```
Monitor multicast;
var   mensaje: integer;
      c, b: signal;
   procedure broadcast(integer m);
     begin
     if (! b.queue()) then
      begin
        mensaje:= m;
        if (esperando >0) then
        c.signal();
        b.wait();
        mensaje:= -1;
      end;
      end;
   procedure termina();
     begin
      esperando:= esperando-1;
      if (esperando >0) then c.signal()
      else if ( b.queue()) then b.signal();
     end;
   procedure fetch(var x:integer);
   begin
     esperando:= esperando+1;
     if (mensaje = -1) then c.wait();
     x:= mensaje
   end;
   begin
     mensaje:= -1;
   end;
```

Nota: Un proceso emisor llama al procedimiento *broadcast (…) p*ara enviar una copia de un mensaje a los procesos receptores que lo hayan solicitado y esperan recibirlo posteriormente. El procedimiento *termina()* es llamado después por cada proceso receptor; el último en recibir desbloquea al emisor.

| F | a) El monitor anterior funciona con cualquier tipo de señal que tenga semántica desplazante

| V | b) Los procesos que llamen al procedimiento *fetch()*mientras hay un mensaje en difusión obtienen una copia del citado mensaje inmediatamente y podrían terminar sin enviar la señal al proceso emisor bloqueado en "b".

| F | c) Las operaciones del monitor anterior sólo funcionarían correctamente si los procesos receptores llaman a la operación fetch() antes de que el emisor llame a broadcast().

| F | d) Si hay un mensaje pendiente de ser recibido (el proceso emisor está esperando), sólo funcionaría bien para el primer proceso receptor ya que el resto no recibiría el mensaje en cualquier secuencia de ejecución.

---

[1] es decir, no se puede impedir, tras la ejecución de una operación *signal(),* que un proceso de la cola de entrada al monitor puede adelantarse al que acaba de ser desbloqueado cuando el monitor quede libre

| | e) La programación del monitor anterior no es correcta con señales desplazantes porque si un receptor ejecuta el procedimiento termina() después de salir del bloqueo y de acabar la ejecución de *fetch()*, la señal que envía al emisor se perdería y éste podría quedar bloqueado indefinidamente cuando tras completar la ejecución del procedimiento *broadcast()*

3 **[2,5]** Cuál de las siguientes opciones sería la correcta para implementar las operaciones *wait(s)* y *signal(s)* de un semáforo general con cualquier tipo de semántica de señales de los monitores (desplazante: SU, SX, SW o no desplazante: SC) y que dicho monitor se pude demostrar que satisface el invariante: {0≤ s} ∧{s= s0 + nV - nP} así como que nunca se produzca robo de señal.

```
Monitor Semáforo;
var s:integer;
c:signal:
procedure P();
begin
 if (s=0) then c.wait()
          else s:= s-1;
end;
```

```
procedure V();
begin
if c.queue() then c.signal()
      else if s=0 then s:= s+1;
end;
begin s:=0; end;
```

a) P():: **if (s=0) then c.wait() else s:= s-1; y** V():: **s:= s+1; c.signal()**
b) P():: **while (s=0) do c.wait() ; s:= s-1; y** V()::**c.signal(); s:= s+1;**
c) P():: **if (s=0) then c.wait() else s:= s-1; y** V():: **if (c.queue()) then c.signal() else s:=s+1;**
d) P():: **if (s=0) then c.wait() else s:= s-1; y** V():: **if (c.queue()) then c.signal() else if (s=0) then s:=s+1;**
e) P():: **while (s=0) do c.wait() ; s:= s-1; y** V()::**s:= s+1; c.signal();**

4 **[3,5]** Una familia de n pajarillos hambrientos comen de un mismo plato que inicialmente contiene F porciones de pienso. Cada uno de los pajarillos come una porción de pienso cada vez y, de forma repetitiva, duerme durante un rato, volviendo después a comer. Los comportamientos de un pajarillo y del padre se pueden modelar, entonces, como sigue a continuación:

```
1  class Pajarillo implements Runnable
2  {
3  private Plato plato; // Compartido por
     todos los pajarillos y el padre pájaro
4  int mi_Id; // Id  de pajarillo (para las
     salidas )
5
6  public Pajarillo(Plato p, int id)
7  { plato = p; mi_Id = id; }
8
9  public void run()
10 {
11 while (true)
12 {
13 System.out.println("Pajarillo " + mi_Id +
     " está hambriento.");
14 plato.obtieneComida();
15 System.out.println("Pajarillo " + mi_Id +
     " está comiendo.");
16 comer();
17 }
18 }
```

```
19 private void comer() {
21 try                                  {
     Thread.sleep((int)Math.round(Math.ran
     dom()*50)); }
22 catch (InterruptedException e) {}
23 }
24 }
25
26 El padre pájaro duerme hasta que el
     plato está vacío, lo completa y vuelve
     a dormir:


27 class Padre implements Runnable {
28 /** El plato que ha de mantener lleno */
29 private Plato plato;
30 public Padre(Plato p)
31 { plato = p; }
32 public void run() {
33 while (true) {
34 plato.completar();
35 System.out.println("Plato
     completado.");
36 }
37 }
38 }
```

Como se puede observar la correcta sincronización entre los pajarillos y el padre se programa dentro de la clase Platos. Se pide:
a) [1.75] Una implementación en pseudo-java de dicha clase (Plato) como un monitor programado en Java.
b) [1.75] Obtener el invariante del monitor que ha de verificar la solución que se proponga para éste y demostrar la corrección de sus procedimientos utilizando las reglas de las operaciones sincronización.
Suponer semántica de señales SU SC

```
Class Plato;
{ private num_porciones = F;
  public void synchronized obtieneComida() {
      while (num_porciones == 0) {
          this.wait(); // si el plato está vacío, esperar a que el padre lo complete
      }
      num_porciones--; // coger 1 porción de pienso del plato
      if (num_porciones == 0)
          notifyAll();
  }
  public void synchronized completar() {
      while (num_porciones > 0) this.wait(); // esperar hasta que un pajarillo
                                             // encuentre el plato vacío
      num_porciones = F; // coloque F porciones y lo pone en el plato
      notifyAll();
  }
```

Padre Pájaro

[P_1] "wait until" un pajarillo encuentre el plato vacío

[P_2] Cocinique F porciones de pienso y las pone en el plato

[P_3] "Informar" a todos los pajarillos hambrientos (esperando poder comer) que hay comida en el plato

Pajarillo

[B_1] dormir durante un rato

[B_2] Comprobar si el plato está (estaba) vacío (num-porcions = 0)

[B_3] Si num-porciones = 0 "wait until" el padre pájaro complete el plato con F porciones de pienso

[B_4] asignar num-porcions := F;

[B_5] coger 1 porción de pienso del plato (num-porcs --)

[B_6] Si el plato se ha quedado vacío (num-porcions = 0) entonces

[B_7] "informar" al padre pájaro

[B_8] Comerse la comida

INVARIANTE

[B_7, P_1] Sale de [P_1] $\leq$ Entra [B_7]

[P_3, B_3] $i : 1..M$ Sale [B_3]$_i \leq$ Entra(P_3) + 1