examen-Tema-2.pdf

user_1587848

Sistemas Concurrentes y Distribuidos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

SCD (22-23). Examen de teoría (resto del tema 2). RESPUESTAS

Grado Informática (grupo C). Conv. ordinaria (eval. continua). 23-Nov-2022 (T2).

Nombre: CARLOS Apellidos: UREÑA

Pregunta 1.1 (3.5 puntos) (un acierto suma 0.5, fallo resta 0.15)

1. En las soluciones de alto nivel para sincronización, cuando un proceso tiene que esperar
   (a) ejecuta un bucle hasta que ocurre lo que espera.
   (b) usa mecanismos de paso de mensajes para esperar.
   (c) no ejecuta absolutamente ninguna instrucción hasta que se supera el tiempo de espera.
   (d) ✔ no ejecuta absolutamente ninguna instrucción hasta que es reanudado.

2. En un monitor con semántica señalar y espera urgente (SU) en un instante dado hay un proceso ejecutando código del monitor, entonces, en ese mismo instante
   (a) no puede haber procesos en la cola de urgentes.
   (b) ✔ puede haber procesos en la cola de urgentes.
   (c) puede haber procesos en la cola de urgentes únicamente si hay procesos en alguna cola condición.
   (d) puede haber procesos en la cola de urgentes únicamente si no hay procesos en la cola del monitor.

3. En un monitor con semántica señalar y continuar, en un momento hay un proceso A bloqueado en una cola `q` y entonces otro proceso B inicia las sentencias `q.signal();print("hola")`, entonces
   (a) ✔ la sentencia `print` se ejecutará antes de que A salga de la cola del monitor.
   (b) la sentencia `print` se ejecutará cuando B sale de la cola del monitor.
   (c) la sentencia `print` se ejecutará cuando B sale de la cola de urgentes.
   (d) nunca se ejecuta la sentencia `print`.

4. Supongamos un problema que ha sido resuelto con un monitor SU (tipo Hoare), y ahora queremos encontrar una solución para ese mismo problema usando semáforos:
   (a) siempre es posible, cada variable condición se traduce en un semáforo y un valor lógico.
   (b) siempre es posible, cada variable condición se traduce únicamente en un valor lógico.
   (c) ✔ siempre es posible, se usa un semáforo para la cola de urgentes y otro para la del monitor.
   (d) no será posible si el monitor tiene más de 3 variables condición.

5. Al derecha aparece un posible algoritmo de exclusión mutua entre dos procesos (`a` y `b` son dos variables lógicas compartidas inicializadas a `false`). Este algoritmo
   (a) cumple la propiedad de espera limitada.
   (b) es correcto.
   (c) no cumple la propiedad de progreso.
   (d) ✔ no cumple la propiedad de exclusión mutua.

```
while b do begin end
a := true ;
  {sección crítica}
a := false ;
```

```
while a do begin end
b := true ;
  {sección crítica}
b := false ;
```

6. Supongamos un algoritmo para exclusión mutua entre dos procesos A y B el cual, en caso de que ambos concurran en el protocolo de entrada, siempre permite pasar al proceso A y deja al B esperando:
   (a) ✔ el algoritmo no cumple la propiedad de espera limitada.
   (b) el algoritmo es correcto.
   (c) el algoritmo no cumple la propiedad de progreso.
   (d) el algoritmo no cumple la propiedad de exclusión mutua.

7. Sea `b` una variable lógica (compartida por dos procesos) que vale `false` en un momento, a partir de ese momento los dos procesos completan cada uno una llamada a `TestAndSet(b)`, así que cada proceso recibe un valor lógico como resultado de dicha llamada:
   (a) ✔ el segundo proceso en iniciar la llamada recibe `true`.
   (b) el primer proceso en iniciar la llamada recibe `true`.
   (c) pueden ser ambos `false`, o bien puede ser uno de ellos `true` como mucho.
   (d) pueden ser ambos `true`, o bien puede ser uno de ellos `false` como mucho.

Pregunta 2 (2.5 puntos)

A continuación aparece un posible algoritmo de exclusión mutua entre dos procesos P y Q. Las sentencias atómicas están etiquetadas (ten en cuenta que las etiquetas P2 y Q2 se refieren a una única lectura atómica de la correspondiente variable lógica).

```
{ declaraciones de variables compartidas }
var c1 : boolean := false ;
    c2 : boolean := false ;
```

```
Process P ;
begin
   while true do begin
      (P1)  c1 := true ;
      (P2)  while c2 do begin end
            {sección crítica}
      (P3)  c1 := false ;
            {resto de sentencias}
   end
end
```

```
Process Q ;
begin
   while true do begin
      (Q1)  c2 := true ;
      (Q2)  while c1 do begin end
            {sección crítica}
      (Q3)  c2 := false ;
            {resto de sentencias}
   end
end
```

Responde a estas cuestiones:

(a) Demuestra que no se cumple la propiedad de progreso, dando como contraejemplo una posible interfoliación (infinita) de las sentencias atómicas etiquetadas, interfoliación que incumple dicha propiedad.

(b) Demuestra que no se cumple la propiedad de espera limitada, dando como contrajemplo una posible interfoliación (infinita) de las sentencias atómicas etiquetadas, interfoliación que incumple dicha propiedad.

Respuesta

Apartado (a). Interfoliación que produce interbloqueo:

P1 - Q1 - **P2 - Q2 -** · · ·

Al final de esta interfoliación ambos procesos ejecutan indefinidamente P2 y Q2 y quedan en el protocolo de entrada. Cualquier interfoliación que comienze con P1-Q1 o Q1-P2 lleva inevitablemente a interbloqueo.

Apartado (b). Interfoliación que no cumple espera limitada:

No hay ninguna interfoliación que no cumpla espera limitada, ya que esa propiedad se cumple siempre. Es un error en el enunciado, este apartado (b) no se ha evaluado (la nota de esta pregunta se consigue únicamente con el apartado (a))

Pregunta 3 (4 puntos)

Queremos escribir un monitor (llamado `T`) para sincronizar dos procesos `P` y `Q`, cada uno de ellos ejecuta las sentencias `S1` y `S2`, respectivamente, en un bucle infinito. El monitor exporta 4 procedimientos llamados `previo1`, `previo2`, `posterior1` y `posterior2`, de forma que el código de los procesos es este:

```
Process P
begin
    while true do begin
        T.previo1();
        S1 ;
        T.posterior1();
    end
end
```

```
Process Q
begin
    while true do begin
        T.previo2();
        S2 ;
        T.posterior2();
    end
end
```

Queremos que se cumplan estos dos requisitos: (a) la sentencia `S2` no podrá ejecutarse nunca un número de veces superior al triple del número de veces que se haya ejecutado `S1`, y (b) las sentencias `S1` y `S2` no se pueden ejecutar ambas a la vez. Responde a estas cuestiones:

(1) ¿ Qué variables permanentes necesitamos, de qué tipos son y para que sirve cada una ?
(2) ¿ Qué variables condición necesitamos, para que sirve cada una y cual su condición asociada ?
(3) Escribe el pseudo-código del monitor.

Respuesta

Necesitamos estas variables permanentes:

- `m`: una variable lógica que nos indique si alguno de los procesos está ejecutando la sentencia `S1` o la `S2`, o bien ninguno lo está.
- `n1`: variable entera, que indica cuantas veces se ha ejecutado `S1`
- `n2`: variable entera, que indica cuantas veces se ha ejecutado `S2`

En lugar de las dos variables enteras, podemos usar una única variable `d`, esa variable indicaría cuantas veces se puede ejecutar `S2`. Las colas condición (con estas variables) son:

- `cQ`: cola donde espera el proceso `Q` si se está ejecutando `S1` o bien si no puede ejecutar `S2` en este momento por ir adelantado más del triple de ciclos respecto de `S1`. La condición es: `not m and n2 < 3*n1`.
- `cP`: cola donde espera el proceso `P` si se está ejecutando `S2`. La condición es: `not m`

Se podría usar una única variable condición, ya que nunca puede ocurrir que ambos procesos estén esperando a la vez. El código del monitor es este:

```
Monitor T ;
 var m  : boolean := false;
     n1 : integer := 0 ;
     n2 : integer := 0 ;
     cP, cQ : condition ;
```

```
procedure previo1()
begin
    if m then
        cP.wait();
    m := true ;
end
procedure posterior1()
begin
    m := false ;
    n1 := n1+1 ;
    cQ.signal();
end
```

```
procedure previo2()
begin
    if m or n1 >= 3*n2 then
        cQ.wait();
    m := true ;
end
procedure posterior2()
begin
    m := false ;
    n2 := n2+1 ;
    cP.signal();
end
```