Programación de Sistemas Concurrentes y Distribuidos 2ª Convocatoria curso 18/19

11 de septiembre de 2019

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Notas previas

- Cada ejercicio debe resolverse en una hoja independiente.
- Todos los ejercicios deben ir acompañados de una clara y concisa explicación de la solución propuesta.

Ejercicio 1 (1.5 ptos.)

Considérese el siguiente esquema de programa concurrente compuesto por un proceso `generador` y N procesos lectores, que comparten una variable (`dato`). El proceso `generador` genera valores que va, sucesivamente, asignando a la variable compartida. Los lectores tienen que coger dichos valores.

```
integer N = 10
float dato
...
Process generador::
   ...
   while true
      // calcular un valor para dato
      ...
      dato := ... //el valor calculado
      ...
   end while
end

Process lector(i: 1..N)::
```

```
   float miDato
   ...
   while true
      ...
      miDato := dato
      ...
   end while
end
```

El ejercicio pide completar el esquema de código propuesto de manera que, utilizando semáforos como medio de sincronización, asegure los siguientes requisitos:

- El proceso generador no sobreescribe un dato que aún no ha sido leído por ningún proceso lector.
- Un valor depositado en la variable `dato` por el proceso `generador` solo puede ser leído por un único proceso `lector`.

Ejercicio 2 (1.5 ptos.)

Considérese un sistema distribuido compuesto por 100 procesos cliente y un proceso coordinador. La misión del coordinador es servir de barrera: cada cliente pide al servidor permiso para pasar y espera a recibirlo del coordinador, ciclo que repite constantemente. Cuando el coordinador acumula 10 peticiones envía el permiso de paso a los 10 clientes. Se pide implementar el sistema usando canales síncronos como medio de comunicación entre los procesos involucrados.

Es preciso describir de forma breve y precisa la estructura de canales usada para resolver el problema. Posteriormente, programar los procesos cliente y el servidor que integran el sistema.

Ejercicio 3 (2.0 ptos.)

Considérese el siguiente esquema de programa concurrente.

```
integer v1 = 4, v2 = 0

process P1
   while(true)
      < await v1 >= 3    //transición t1
          v1 = v1-3
      >
      < v2 = v2+1 >      //transición t2
   end while
end process

process P2
   while(true)
      < await v1 >= 1    //transición t1P
```

```
            v1 = v1-1
        >
        < await v2 >= 1     //transición t2P
            v2 = v2-1
        >
        < v1 = v1+4 >       //transición t3P
    end while
end process
```

El ejercicio pide:

- Diseñar un modelo red de Petri para el programa
- Obtener el grafo de estados alcanzables del modelo propuesto
- Razonando sobre el grafo obtenido dar respuestas convincentes a las siguientes cuestiones:
  - Pruébese que el sistema no tiene problemas de bloqueo.
  - ¿Es el programa equitativo?
  - Pruébese la siguiente relación invariante: $0 \leq v1 + v2 \leq 4$

Ejercicio 4 (2.0 ptos.)

Un programa concurrente consta de M procesos que se sincronizan entre sí a través de un tipo especial de barrera. El funcionamiento de esta barrera es el siguiente. El primer proceso que llega establece el número de procesos (parámetro *numProcesos*) que deben alcanzar la barrera antes de poder pasarla. Obviamente, este número debe ser menor o igual que el total de procesos del programa. Cuando otros *numProcesos*-1 procesos alcancen la barrera, todos los procesos involucrados en la sincronización pueden continuar su ejecución. A partir de ese instante, se vuelve a repetir el funcionamiento descrito.

La barrera se implementa mediante el monitor `Barrera` con la operación `sincEnBarrera`, de acuerdo con el esquema de programa que se muestra debajo. El parámetro *numProcesos* sólo será utilizado cuando el proceso que invoca la correspondiente operación es el primero que llega a la barrera. En el resto de los casos es ignorado.

Se pide programar el monitor *Barrera* y los correspondientes procesos, así como describir de forma breve y justificada las decisiones de diseño adoptadas para garantizar el correcto funcionamiento del programa. No se evaluarán aquellas soluciones que no estén debidamente justificadas.

Nota: la solución es más simple asumiendo la política clásica de reanudación en monitores ("Immediate Resumption Requirement"), en que "E <S <W".

```
Monitor Barrera::
   ...
   operation sincEnBarrera(integer numProcesos)
      ...
   end
```

```
end
//--------------------------------------
Process P(i:1..M)::
   integer numProcesos
   integer tiempo
   ...
   while true
     tiempo := ... //un valor aleatorio
     sleep(tiempo)
     numProcesos := ... //un valor aleatorio de [1,M-1]
     Barrera.sincEnBarrera(numProcesos) //Barrera de sincronización
     ...
   end while
end
```

Ejercicio 5 (3.0 ptos.)

Un sistema distribuido está formado por 100 procesos *Cliente* que compiten por tener acceso a una zona restringida. El problema está en que, como máximo, 10 procesos pueden tener acceso simultáneo. Un proceso que pretenda entrar cuando la zona está completa deberá esperar. Por otro lado, una vez que un proceso ha entrado no puede salir cuando quiera, sino cuando un proceso *Gestor* se lo indique. El gestor, en cada iteración, duerme un rato, espera a que haya al menos dos procesos dentro de la zona y les indica que pueden salir. Desde el instante en que el gestor empieza a comprobar si hay al menos dos peticiones hasta que todos los procesos han salido no podrá entrar en la zona ningún nuevo proceso.

Se pide programar el sistema distribuido descrito y coordinar la interacción y sincronización de los procesos involucrados por medio de *Linda*. La solución deberá contener una descripción detallada del protocolo diseñado para la interacción entre los procesos y el código de los procesos *Cliente* y *Gestor* (conforme la sintaxis presentada en clase).