Programación de Sistemas Concurrentes y Distribuidos 2ª Convocatoria curso 13/14

4 de septiembre de 2014

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Ejercicio 1 (2.5 ptos.)

Consideremos un sistema con un conjunto de procesos distribuidos en el que cada proceso realiza una operación específica. Dada la naturaleza de las tareas, es posible que su ejecución falle. Consideremos el caso en que necesitamos que cada uno de los procesos realice su operación, pero bajo un contexto de transacción: es indispensable que todos realicen su operación, pero que ninguno falle. Es lo que se suele denominar una ejecución "todo o nada": si alguno falla, ninguno debe llevar a cabo la operación (o, lo que es lo mismo, los que la hubieran ejecutado con éxito deben deshacerla).

Una posible solución al problema es el *two-phase commit protocol* (2PC). Es un caso de algoritmo de consenso distribuido en que hay un proceso coordinador de transacción, TM, y un conjunto de procesos participantes, TP. El protocolo se ejecuta en dos fases: la *Voting phase* y la *Commit phase*, que se describen a continuación (se trata de una versión simplificada, sin `timeouts`).

- Voting phase
  - El coordinador envía un mensaje `query-to-commit` a todos los participantes, y espera hasta recibir un mensaje de respuesta de cada uno de ellos. Por su parte, cada participante ejecuta su operación.
  - Cada participante, una vez recibido el mensaje `query-to-commit`, ejecuta su operación y envía al coordinador un mensaje `Yes` con un voto positivo en

caso de que la operación se haya ejecutado con éxito, y un voto `No` en caso de problemas en la ejecución

- Commit phase
  - Si el coordinador recibe en la fase anterior un voto `Yes` de cada participante:
    - El coordinador envía un mensaje `Commit` a cada participante, y espera un mensaje de acuse de recibo `Ack` de cada uno de ellos
    - Cada participante envía un mensaje `Ack` al coordinador
    - Una vez que el coordinador recibe todos los mensajes `Ack` da por finalizada la transacción
  - Si el coordinador recibe en la fase anterior un voto `No` de algunos de los participantes:
    - El coordinador envía un mensaje `Rollback` a todos los participantes, y espera un mensaje de reconocimiento `Ack` de cada uno de ellos
    - Cada participante que votó `Yes` deshace la operación
    - Cada participante envía un mensaje `Ack` al coordinador
    - Una vez que el coordinador recibe todos los mensajes `Ack` da por abortada la transacción

El ejercicio pide completar el código que se muestra a continuación, y que implementa los procesos coordinador y participantes en un 2PC, usando comunicación síncrona. Hay que sustituir los comentarios `Voting phase` and `Commmit phase` por el código que se considere oportuno.

```
constant integer N := ... //N > 2
channel of String array[1..N] C

Process TM::
   integer numPos //num de resp. 'Yes' en la fase Voting

   //Voting phase
   //Commit phase

   if numPos=N
      write('Transacción␣finalizada')
   else //es 'Rollback'
      write('Transacción␣abortada')
   end if
end Process

Process TP(i: 1..N)::
   //Para ejecutar su operación, se invoca la función
   //execute_operation_i(), que devuelve 0 si ha habido fallo
   //y 1 si la ejecución es correcta

   //Para deshacer la ejecución de la operación, se invoca
   //el procedimiento undo_operation_i()

   //Voting phase
   //Commit phase

end Process
```

Ejercicio 2 (2.5 ptos.)

Se ha aplicado el algoritmo de los Generales Bizantinos a un escenario con 4 generales (Basilio, Zoe, León y Juan). Uno de estos generales es traidor y genera fallos bizantinos en segunda vuelta con el propósito de provocar inconsistencias en las decisiones de los generales leales. La figura **??** muestra la estructura de datos de Basilio, uno de los generales leales, después de ejecutar el algoritmo.

Se pide:

- En base a la estructura de datos de Basilio, ¿qué general es el traidor? Responde de manera razonada y concisa a la pregunta.
- Describir un ejemplo concreto de aplicación del algoritmo que tenga como resultado la estructura de datos anterior de Basilio. La descripción debe detallar los mensajes concretos que intercambian los generales en primera y segunda vuelta, así como sus correspondientes estructuras de datos resultantes (para cumplimentar esta última información se proporcionan adjuntas al enunciado las estructuras de datos vacías

Basilio					
General	**Plan**	**Reported by**			**Majority**
		Zoe	León	Juan	
Basilio	A	—	—	—	A
Zoe	A	—	A	R	A
León	R	R	—	A	R
Juan	A	A	A	—	A
Majority					A

Figura 1: Información de Basilio tras la segunda vuelta

de los generales involucrados).

- Sobre el escenario planteado como solución al apartado anterior, ¿cuál es la decisión final de los generales leales? ¿Son consistentes sus decisiones? Responde de manera razonada y concisa a estas preguntas.

Ejercicio 3 (2.5 ptos.)

Considérese el siguiente programa concurrente:

```
                    integer s := 3

process UNO::       process DOS::       process TRES::
   loop forever        loop forever        loop forever
      <await s>0          <await s>1          <await s>2
         s := s-1            s := s-2            s := s-3
      >                   >                   >
      //SC1               //SC2               //SC3
      <s := s+1>          <s := s+2>          <s := s+1>
   end loop            end loop            end loop
end process         end process         end process
```

El ejercicio pide:

1. Dar un modelo red de Petri correspondiente al programa
2. A partir del modelo anterior, construir su espacio de estados
3. Basándonos en el espacio de estados del modelo, responder razonadamente a las siguientes preguntas

   a) ¿Puede el programa llegar a bloquearse totalmente (es decir, llegar a un estado en el que que ninguna instrucción se pueda ejecutar)? En caso afirmativo, hay

que dar una secuencia de ejecución que lleve a bloqueo total. En caso negativo, hay que razonar por qué no puede bloquearse.

b) ¿Puede el programa llegar a bloquearse parcialmente (es decir, llegar a un estado en el que si bien a partir de él siempre puede ejecutarse alguna instrucción, hay algún proceso que nunca va a poder ejecutar ninguna de sus acciones, independientemente del tipo de "scheduler")? En caso afirmativo, hay que escribir una secuencia de ejecución que lleve a bloqueo parcial. En caso negativo, hay que razonar por qué no puede bloquearse.

c) ¿Existe alguna historia no equitativa (entendiendo que es una ejecución infinita en que a partir de un momento determinado algún proceso que desea intervenir no llega a hacerlo)?

d) Pruébese que se cumple el invariante: "Mientras TRES está en SC3, ni UNO está en SC1 ni DOS está en SC2".

e) Pruébese la siguiente propiedad: "Puede ocurrir que mientras que UNO está en SC1, DOS esté en SC2".

Ejercicio 4 (2.5 ptos.)

Para mejorar las condiciones de estudio de sus alumnos, un centro universitario ha dividido su biblioteca en 10 salas de estudio de pequeñas dimensiones. Cada sala puede ser utilizada por un único alumno, o por dos si ambos están cursando la misma titulación. En este último caso, los dos alumnos que compartan sala no tienen por qué comenzar o finalizar su estudio al mismo tiempo, ni esperarse entre ellos.

El centro tiene actualmente matriculados 100 alumnos y oferta 3 titulaciones diferentes. El comportamiento de un alumno es siempre el mismo: solicita entrar en una sala de estudio, estudia un tiempo aleatorio, finaliza su estudio y sale de la sala. Luego dedica un tiempo aleatorio a alguna actividad de ocio. Cada alumno tiene un identificador de estudiante único y está matriculado en una sola titulación.

Por otro lado, para mantener las salas en perfecto estado, una persona de limpieza se encarga de su mantenimiento. Las salas se limpian en orden, una detrás de otra, y únicamente cuando están desocupadas. Si debe procederse a la limpieza de una sala y ésta está ocupada, la persona encargada tendrá que esperar. Obviamente, mientras que una sala está siendo objeto de limpieza no estará disponible para los alumnos.

Se pide escribir un programa concurrente, utilizando la notación algorítmica presentada en clase, que simule el funcionamiento del sistema descrito. En primer lugar se deberá definir un monitor que controle el acceso a las salas de estudio conforme a las condiciones descritas y la sincronización necesaria entre los alumnos y el responsable del mantenimiento. Posteriormente, se deberán diseñar un proceso alumno y un proceso personal de mantenimiento, conforme a las condiciones previamente descritas.

```
constant integer nAl := 100   //100 alumnos
constant integer nEst := 3    //3 carreras
constant integer nSalas := 10 //10 salas

monitor controlAcceso
     operation titulacion(integer id): integer
        //Pre:  1<=id<=nAl
        //Post: titulacion = valor entre 1..nEst
        //      en que el alumno id está matriculado
        //Com:  Se supone ya implementada
     end operation

     ...
end monitor

process alumno(id: 1..nAl)::
     ...
end process

process mantenimiento()::
     ...
end process
```