Programación de Sistemas Concurrentes y Distribuidos 2ª Convocatoria curso 17/18

11 de septiembre de 2018

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Ejercicio 1 (2.5 ptos.)

Considérese un programa en el que 10 procesos tienen que calcular, de manera concurrente, el máximo de un matriz 40x40 de números reales. El ejercicio pide completar el esquema de programa que se muestra debajo, de manera que:

- Cada proceso busca un máximo parcial de una submatriz, trabajando todos los procesos con el mismo número de datos
- Una vez que los diez procesos han realizado su trabajo, el proceso de identificador 1 muestra por la salida estándar el máximo global
- Los aspectos de sincronización se deben resolver mediante monitores

```
constant integer N := 40
// array de 40x40 doubles
double array[1..N,1..N] D := ... //ya inicializado
...
process ComputeMaxLocal(i::1..10)
  ...
end process
```

Ejercicio 2 (2.5 ptos.)

En este ejercicio se plantea el mismo problema que en el ejercicio anterior, pero para un sistema distribuido compuesto por un proceso coordinador y 10 procesos trabajadores, que utilizan Linda como medio para la sincronización. Asumimos que todos los trabajadores disponen de una copia idéntica de la matriz, ya inicializada. El coordinador va solicitando a los trabajadores que calculen el máximo (parcial) de las distintas filas de la matriz. Una vez se hayan procesado todas las filas, el coordinador muestra por su salida estándar el máximo global de la matriz. Dada la naturaleza distribuida del sistema, con procesos que, posiblemente, se ejecutarán a diferentes velocidades, no es admisible imponer que los trabajadores resuelvan el mismo número de filas.

Es preciso que los procesos terminen de una manera ordenada: tanto los trabajadores como el coordinador mueren una vez terminan su trabajo, y el espacio Linda queda en el mismo estado que tuviera antes de arrancar los procesos.

Ejercicio 3 (2.0 ptos.)

Considérese el esquema de sistema distribuido que se muestra debajo. Está compuesto por dos procesos, *P1* y *P2*, que van a suministrar, uno a uno, sus datos al proceso *merge*, utilizando una red de canales síncronos (a definir). El proceso *merge* va tomando, de manera sucesiva, un dato de P1 y otro de P2, calcula la media y la muestra por su salida estándar. Una vez que los datos de uno de los dos procesos se han consumido (las constantes N de *P1* y *P2* no tienen necesariamente el mismo valor) toma y muestra los datos del otro, hasta que este también finalice.

```
...

process P1()
   const int N := ...        //ya inicializado
   double array[N] D := ... //ya inicializado
   ....
end process

process P2()
   const int N := ...        //ya inicializado
   double array[N] D := ... //ya inicializado
   ....
end process

process Merge()
   ....
end process
```

Ejercicio 4 (3.0 ptos.)

En un pequeño pueblo del Pirineo viven 20 personas y un peluquero. Este último es muy sociable y ha convertido su barbería en un punto de encuentro al que acuden las personas del pueblo cada cierto tiempo. La barbería tiene un sillón para cortar el pelo y 5 sillas para que esperen los clientes. Si no hay clientes, el peluquero tiene la costumbre de dormir una siesta. Si está dormido, el primer cliente que llega a la peluquería le tiene que despertar. El tiempo que cuesta cortar el pelo varía dependiendo de lo animada que esté la conversación entre el peluquero y su cliente. Si durante el servicio llegan nuevos clientes, estos se irán sentado en las sillas libres, mientras haya disponibles, y esperarán a ser llamados en orden de llegada por el peluquero para pasar al sillón. Una vez en el sillón, cuando tenga el pelo cortado, el barbero le hará salir del establecimiento.

El objetivo del ejercicio es diseñar un sistema concurrente que simule el funcionamiento de la peluquería. El sistema consta de dos tipos distintos de procesos: el proceso *Persona* y el proceso *Peluquero*. El comportamiento de estos procesos corresponde con el descrito en el párrafo anterior.

Se pide:

- Plantear una primera versión del diseño del sistema basada en la instrucción *await*.
- Programar el diseño anterior utilizando semáforos como primitiva de sincronización.

```
boolean dormido := true //estado inicial del peluquero
...

process Peluquero()
   ...
   loop forever
     ...
   end loop
end process

process Cliente(idC:1..20)
   ...
   loop forever
     ...
   end loop
end process
```