Programación de Sistemas Concurrentes y Distribuidos

2ª Convocatoria curso 16/17 (11/09/2017)
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Notas

- Hay que resolver los ejercicios en hojas separadas.
- Hay que poner sobre la mesa un documento de identidad del alumno.
- No se considerarán las soluciones que solo consten de código y no estén explicadas de forma clara y concisa.

Ejercicio 1 (2.0 ptos.)

Considérese el código, en la notación algorítmica usada en clase, que se muestra en el Anexo I. El ejercicio pide:

- Presentar un modelo red de Petri que refleje el comportamiento del programa concurrente
- Obtener el grafo de estados alcanzables del modelo propuesto
- Razonando *sobre el grafo obtenido* dar respuestas convincentes a las siguientes cuestiones:
  - ¿Presenta el programa problemas de bloqueo? En caso afirmativo, enumerar esos estados. Para cada uno de estos hipotéticos estados se pide estimar cuántas posibles ejecuciones lo alcanzan, y dar una de ellas.
  - Si nos centramos en las historias (infinitas) que no llevan a bloqueo, ¿se pueden considerar equitativas? Demostrar la corrección de la respuesta dada.

- Demostrar que se cumple el siguiente invariante: "Siempre que P2 ha ejecutado sus dos primeras instrucciones, pero no la tercera, los procesos P1 y P3 tienen que permanecer en su estado inicial".

Ejercicio 2 (2.5 ptos.)

Un sistema distribuido consta de cuatro procesos que deben encontrar el máximo de una matriz de enteros de dimensión 100x100. Cada proceso tiene un copia local de la matriz completa. Cada proceso calcula el máximo local de una fila y actualiza consecuentemente el máximo global calculado hasta el momento. Este comportamiento se repite mientras la matriz no haya sido procesada por completo.

No todos los procesos son igual de eficientes: dependiendo de varios factores, el tiempo de procesar una fila de la matriz puede variar bastante entre unos y otros. Por lo tanto, no se consideran válidas soluciones en las que se haya prefijado estáticamente las filas que un proceso debe analizar.

Se pide desarrollar el sistema descrito coordinando la actividad de los procesos meidante Linda. La solución se debe programar con la notación utilizada en clase y debe cumplir los siguientes requisitos:

- El sistema debe ser lo más eficiente posible desde el punto de vista del procesado.
- El sistema consta únicamente de los cuatro procesos encargados de la búsqueda del máximo.
- Una vez encontrado el máximo de la matriz, todos los procesos deben finalizar de forma correcta y uno de ellos debe mostrar el resultado por su salida estándar.

Ejercicio 3 (2.75 ptos.)

Una gasolinera tiene un túnel de lavado de coches, controlado por el proceso *Control_túnel*, y un aparcamiento (que podemos asumir de tamaño infinito). Cuando un usuario quiere lavar su vehículo, lo deposita en el aparcamiento con las llaves puestas y se va al bar, hasta que le avisan de que su coche está listo momento en el que lo retira. La gasolinera tiene un único empleado. Su trabajo consiste en ir al aparcamiento a por el siguiente coche, meterlo en el túnel, activar el proceso que controla el lavado y sentarse hasta que este proceso finalice. Avisa entonces al dueño del coche para que se lo lleve. Este comportamiento lo repite durante toda su jornada laboral. El tiempo que cuesta lavar un coche depende de lo sucio que esté.

Se pide implementar un programa concurrente, con la notación algorítmica usada en clase, que simule el funcionamiento del sistema anterior. Este tendrá 100 procesos *Usuario* que cada cierto tiempo llevarán su coche a la gasolinera para que sea lavado, un

proceso *Empleado* y el proceso *Control_túnel*. Las cuestiones relativas a la concurrencia y sincronización de los procesos se programarán por medio de un monitor.

Ejercicio 4 (2.75 ptos.)

Consideremos un sistema con dos tipos de recursos, R_1 y R_2, habiendo 10 unidades de cada uno de ellos. Consideremos también dos tipos de procesos, cuyos esquemas son los siguientes:

```
process P(i::1..10)

  while(true)
     //reserva los recursos 2R_1+R_2
     //usa los recursos
     //libera los recursos
  end while
end process

process Q(i::1..15)

  while(true)
     //reserva los recursos R_1+2R_2
     //usa los recursos
     //libera los recursos
  end while
end process
```

El ejercicio pide desarrollar el programa, con la notación utilizada en clase, que implemente el sistema descrito. La coordinación se hará a través de un proceso servidor con comunicación síncrona. A la hora de volver a dar recursos que son devueltos, el coordinador dará preferencia a los procesos que sean del mismo tipo que el proceso que devuelve los recursos, pero en ningún caso puede no dar recursos disponibles. La atención a los procesos cliente se debe hacer en orden de llegada (política FIFO).

Anexo I

```
integer s1 = 3, s2 = 1

process P1
  while(true)
     < await s1>1
         s1 = s1-2
     >
     < await s2>0
         s2 = s2-1
     >
     < s1 = s1+2
       s2 = s2+1
     >
  end while
end process

process P2
  while(true)
     < await s1>=3
         s1 = s1-3
     >
     < await s2>0
         s2 = s2-1
     >
     < s1 = s1+3
       s2 = s2+1
     >
  end while
end process

process P3
  while(true)
     < await s2>0
         s2 = s2-1
     >
     < await s1>=2
         s1 = s1-2
     >
     < s1 = s1+2
       s2 = s2+1
     >
  end while
end process
```