EJERCICIOS TEMA 3 PROGRAMACION CONCURRENTE

FECHA REALIZACIÓN:
NOMBRE:
NOTA:

1. **Ejercicio 3.5 del libro de G. Andrews.** Algunas máquinas tienen una instrucción para intercambiar el valor de dos posiciones de memoria de manera atómica:

   ```
   Exchange(int var1,int var2)
    < int temp; temp=var1; var1=var2; var2=temp; >
   ```

   Utilizando la instrucción `Exchange`, se pide desarrollar una solución al problema de la sección crítica que use una variable global *lock* inicializada a 0. La solución no ha de ser justa.

2. **Ejercicio 3.10 de libro de G. Andrews.**

   - Modifica la solución de *alto nivel* del algoritmo del ticket (es decir, la que utiliza "<..>" para representar la ejecución atómica) para "n" procesos ($n < 100$) que garantice que las variables *next* y *number* no desbordan.
   - Modifica la solución de *bajo nivel* del algoritmo del ticket (donde no está permitido utilizar "<..>") para "n" procesos ($n < 100$) de manera que las variables *next* y *number* no desborden. Asume que dispones de la instrucción *fetch-and-add*.

3. **Barreras.** Implementa el proceso coordinador para que la barrera que utilizan los siguientes n procesos Worker funcione correctamente:

   ```
   int arrive[n] = ([n] 0);
   int continue[n]=([n] 0);

   process Worker[i=1 to n] {
     while (true)
       { // iteracion proceso i
         arrive[i] = 1;
         while (continue[i]!=1);
         continue[i]=0;
       }
     }
   ```

   ```
   process Coordinador{
   ```

4. **Barreras.** Implementa una "barrera de diseminación" reutilizable para 14 procesos. Para ello, implementa el método `void barrera(int id)` que es invocado por cada proceso con su identificador `id` (de 1 a 14). Puedes utilizar la instrucción `await` para especificar puntos de espera. Muestra la declaración e inicialización de todas las variables que necesites, indicando si son variables globales.