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. **Uso de barreras en algoritmo de listas.** Considera el siguiente programa secuencial de complejidad lineal para encontrar la posición final de una lista simple enlazada con al menos dos elementos (los enlaces entre los elementos de la lista se almacenan en el array *link* y *head* apunta al primer elemento, los datos se almacenan en un array *data* que no vamos a utilizar):

   ```
   process sequencial(int head, int[] link)
   {  i = head; bool found=false;
      while (not found)
      {
      if  (link[i]!=null)
        i=link[i];
      else found=true;
      }
      return i;
   ```

   El valor de link[i] para el último elemento de la lista está representado por la constante entera *null* (los elementos del array que no forman parte de la lista también son *null*). Como ejemplo de la ejecución, dado un array *link* (indexado del 0 al 5 y siendo $head = 0$) con el siguiente contenido $link = [2, 3, 1, 4, null, null]$, el resultado de la ejecución será: $i = 4$.

   Implementa un algoritmo de datos paralelos utilizando la técnica de "doblar la distancia" que cree "n" procesos concurrentes y cada uno de ellos encuentre la posición final de la lista en complejidad logarítmica.