Erlang. Concurrencia.

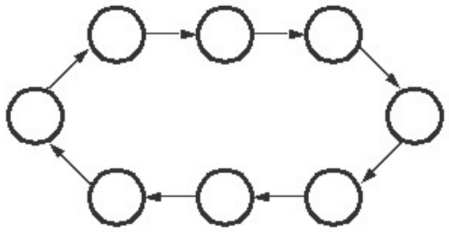
Práctica 2

Deben entegarse via Campus Virtual solo los ejercicios 1 a 4

1. Implementar un módulo `clock`, tal como el que se vio en clase, pero corrigiendo el comportamiento inesperado de dicha implementación (véanse pags 54-55 en las transparencias).

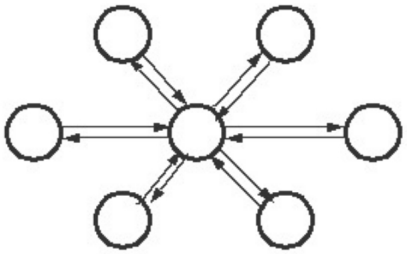
2. Implementar una versión concurrente de la ordenación rápida de Hoare (quicksort) para listas.

3. Escribir un programa que arranque N procesos conectados en anillo (cada uno con el siguiente y el último con el primero) tal como muestra el dibujo.



   El primero de estos procesos debe enviar un mensaje que circule M veces a través del anillo y después los procesos deben terminar. La llamada incial de módulo será *ring:start(M, N, Msg).*, done M es el número de vueltas, N el número de procesos y *Msg* el mensaje a enviar.

4. Escribir un programa que arranque N procesos conectados en estrella tal como muestra el dibujo.



   El proceso central debe envíar un mensaje M veces a cada uno de los demás y después terminar todos.

5. En este ejercicio queremos simular un grupo de N jugadores de baloncesto pasándose una pelota entre ellos de manera aleatoria. Hay además un entrenador encargado de arrancar el juego (hacer el primer pase) y de terminarlo. Cada uno de los jugadores vendrá implementado mediante un proceso registrado `player1, player2,...,playerN` (tenerlos registrados facilitará la implementación), que ejecutará el código de simulación de recepción, envío de la pelota (paso de mensajes) y quedar a la espera de un nueva recepción. Además debe notificar en pantalla cada pase que hace (un pase es un par emisor-receptor).

   Para hacer una simulación más realista se puede retardar cada pase unos milisegundos utilizando timeouts. El entrenador, a su vez, se implementará mediante otro proceso (no es necesario registrarlo) que hace el primer pase y que para el juego con el mensaje `stop`. La parada del juego conlleva la destrucción de todos los procesos asociados a los jugadores y el *des*-registro de los nombres correspondientes.

   Notas: para seleccionar aleatoriamente un jugador puede importarse la función `uniform` de la librería `random`. Para generar los nombres de proceso `player1, player2`, etc, serán útiles las funciones `list_to_atom` y `integer_to_list`. Un proceso puede terminarse con la llamada `exit(P,kill)` (consultar manuales).

6. Escribir una función `start(AnAtom, Fun)` que registre el proceso `spawn(Fun)` con el nombre `AnAtom`. Moficiar la función para:

   - (difícil) Asegurarse de que el programa funciona correctamente en el caso de que dos procesos paralelos evalúen `start/2` simultáneamente. En ese caso se debe garantizar que solo uno de los procesos tiene éxito y el otro falla.
   - (Más difícil) Asegurarse además de que si dos procesos paralelos evaluan simultáneamente `start(AnAtom,Fun1)` y `start(AnAtom,Fun2)` respectivamente, solo uno tiene éxito y el otro falla **sin llegar a crear el proceso asociado a la función `Fun` que corresponda**.