Programación Concurrente y de Tiempo Real

PRÁCTICA 4

Mª Teresa Supervielle Sánchez

1. Ejercicio 1: tryThree.java y tryFour.java

Una vez leída la documentación indicada e implementado y ejecutado los dos algoritmos, podemos observar que, en el primer algoritmo, el tryThree.java, entra aposta en *Livelock*, eso significa que el algoritmo se sigue ejecutando en un bucle infinito, sin llegar a hacer nada útil. Por otro lado, el segundo algoritmo, tryFour.java si se ejecuta correctamente dando como resultado 0, es decir, lo esperado.

2. Ejercicio 2: algDekker.java

El algoritmo de Dekker permite a dos procesos compartir un recurso **(n)** sin conflictos, para la implementación de la misma se hace uso de un variable **(turn)** la cuál elige cuál de los dos procesos es el que accederá a la Sección Crítica. Por lo tanto, este algoritmo, en la mayoría de sus ejecuciones, da como resultado lo esperado, 0 en este caso.

3. Ejercicio 3: algEisenbergMcGuire.java

El algoritmo de Eisenberg-McGuire, inspirado en el algoritmo original de Dijkstra, fue el primer algoritmo conocido en solucionar el problema de la Sección Crítica. Para ello, utilizó un indicador **flag** con el cuál sabía en que parte se encontraba el proceso: si no quiere entrar en la Sección Crítica ni está en ella, si el proceso quiere entrar en ella o si el proceso ya se encuentra dentro de la Sección Crítica. El resultado tras ejecutar el algoritmo es un bucle infinito, ya que sólo se utiliza uno de los dos hilos.

4. Ejercicio 4: algHyman.java

El algoritmo de Hyman se considera incorrecto, ya que aunque a primera vista, puede parecer que está bien; podría darse la posibilidad de que uno de los procesos quiera entrar, pero sea turno del contrario y este no este preparado para entrar en la Sección Crítica, haciendo que se desincronicen las entradas de los procesos en las siguientes iteraciones. Por lo tanto, los resultados dados son muy dispares a los esperados.