UNIVERSIDAD DE MÁLAGA

Dpt. Lenguajes y Ciencias de la Computación

Systems Programming and Concurrency

(*Programación de Sistemas y Concurrencia*)

Lab nº 6.1. Semaphores

1. In a factory processing plant there are three sensors that measure the levels of temperature, humidity and light. When measurements from the three sensors have been retrieved, a worker device uses them to start several tasks.

   Must be noted that the device cannot start its tasks until every sensor has retrieved a measurement (from now on, a sample); and a sensor cannot obtain a new sample until the device finishes its tasks. This process is repeated indefinitely, so when the device finishes its tasks, it waits again for new samples to be available.

   Develop such a system using Java semaphores. Both the worker device and the sensors must be modeled as threads (so the system must be modeled with 4 threads). As well, both the processes of taking samples and the device process of performing tasks take a random time to be executed. At the very beginning, the whole process starts with the sensors taking samples.

2. In an assembly line there is a main robot whose task is to place 3 different types of products on the conveyor belt. Other robots take the products from the belt to package them. Each of these secondary robots is specialized on a single type of product (1, 2 or 3) whereas it skips the other ones. We want to control the total amount of packaged products independently of their types.

   We have to model this system by means of semaphores following the next constraints:
   - Each robot must be modeled as a thread (1 main robot putting and 3 secondary packagers taking, one for each type of product).
   - The products are put one at each step, and only can be placed into empty places (you can consider the conveyor belt as a circular belt with N places, where any place can hold any type of product). If there is no empty place, then the main robot has to wait for a place to become available.
   - The type of the secondary robots (1, 2 or 3) is specified when created.
   - Each packager robot looks the belt for a product of its type. If there is anyone, then it takes it (one at each time) and frees the place it was placed on, so it becomes available to put new products. If there is not anyone, then it waits for new products to be placed.
   - All the threads work at the same time.
   - The assembly line works endlessly.
   - When a packager robot processes a product, the total amount of packaged product is increased and a message is displayed.

3. We want to simulate a simplified airport, its traffic in particular, following the next constraints:
   - There is a single runway which must be shared by all planes.
   - Planes can close to the airport from the north or from the south. Therefore, to control the traffic two threads must be used one for each way of approximation.

- There are N planes (N threads), and each of them requires to land on the airport at irregular intervals (from a random way). The process for a plane to land is:
    - The plane asks for landing permission to the controller in charge of its way of approximation (north or south). Then it waits (perhaps using a semaphore) until its request is satisfied (it is given permission).
    - Controllers accept the requests one at each time. When a new request is received, they must wait until the runway becomes available and then they call the plane to give the permission. When the plane finishes the landing operation, the controller is informed in order it to free the runway. If there is no request then the controller is idle (must go to sleep).
    - The landing operation is simulated by means of a random time. Throughout this time the runway is unavailable.

Messages must be displayed both when a plane starts landing and when it finishes such a operation.

4. Single-lane bridge. In this problem we want to simulate a bridge so narrow than the cars can run in a single way. You have to develop a solution for this problem, following the next constraints:
    - There are N cars (N threads) which try to pass through the bridge from left to right or the other way.
    - At each moment, the bridge may be empty or in use by cars which run in a single way (left to right or right to left). To simplify the problem, you can assume that the bridge can hold any number of cars at the same time.
    - The cars works the same way all the time: wait until can enter into the bridge, run through the bridge and leave the bridge (to simplify, we can assume that each car works always from left-to-right or right-to-left). After leaving the bridge, a car waits a random time before trying again to cross the bridge. In addition, there is no need to take into account that the first cars to enter must be the first cars to leave.

You can develop two different solutions for this problem:

- Unfair version where one side of the bridge can wait forever while a continuous line of cars is crossing from the other side.
- Fair version where starvation is prevented. This can be achieved in several ways:
    - By controlling that the number of cars crossing from one side it is already “enough”.
    - By controlling that the number of cars waiting on one side does not overflow a predefined constant value.

The program has to work fine in any case. Neither busy waiting nor additional threads are allowed.

5. Implement the Producer/Consumer problem using several producer and several consumers and using full-handled data. Full-handled data means that each data/item (placed in the buffer by any producer) must be consumed once by every consumer before it could be removed from the buffer. The rest of the behaviour is as in the producer/consumer problem already studied in class.
   For example, let’s say the buffer contains the items [1,2,3,4] and there are 3 consumers, where each of them has consumed the next items:
   C1: 1, 2
   C2: 1
   C3: 1, 2, 3

A cell of the buffer is not freed until the item it holds is consumed by C1, C2 and C3. Following the example, we may see every consumer has already consumed the item 1 so its place in the buffer can be reused to store another item. On the other hand, the item 2 has been consumed by C1 and C3 only, so its place cannot be reused yet.