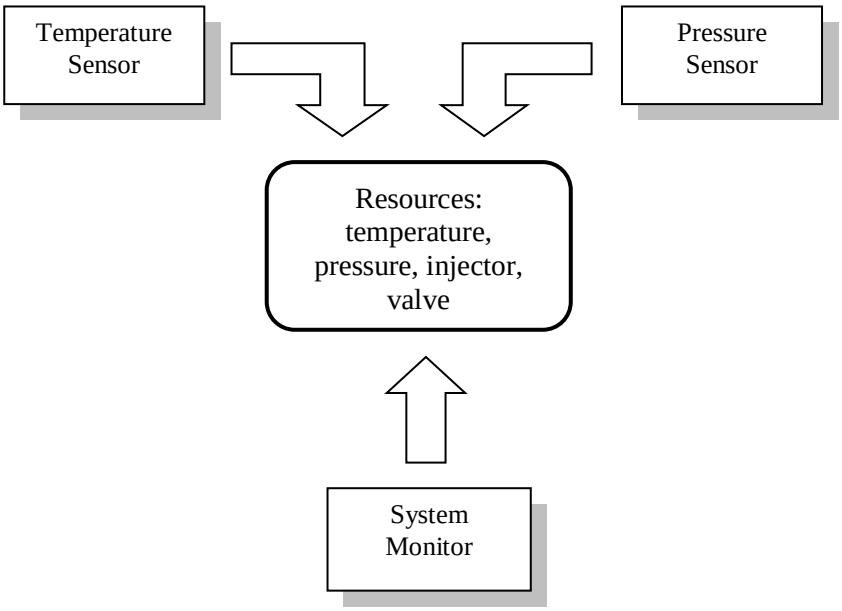
Systems Programming and Concurrency

(*Programación de Sistemas y Concurrencia*)

Lab nº 6.2. Monitors

1. Let's develop a real time software to manage a System for Controlling Pressure and Temperature. To do this, the next tasks are executed concurrently at random intervals:



Control of monitoring requests. This task is executed once every second, retrieves shared resources: pressure, temperature, injector and valve states, and displays all this information on console.

Control of Temperature. This task is executed once every 1000 ms. In each execution it takes 1000 temperature samples and calculates the average of them. In addition, the system review each sample taken and, if exceeds 100º C, it injects cold air into the circuit (this is simulated by a simple message in console). When the temperature reaches good values again (under 100º C) the injection of air stops. Of course, it makes no sense to activate the cold air injection if it is already activated (i.e., it can be activated when deactivated).

Control of Pressure. This task is executed periodically every 2000 ms. Because pressure changes slowly we need to take only 650 samples in each execution. The purpose of this control is similar to the previous one but, instead of cold air injection, this task opens and closes an exhaust valve to control the pressure when it reaches a value over 1000KPa.

2. 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.

3. We have a system composed by three **Smokers** threads who spend their time making cigars by hand and smoking them. To make a cigar they need three resources: **tobacco**, **vanilla** and **matches**. The available **amount of each ingredient is endless** but each ingredient is hold by a single smoker only, i.e. one smoker has all the **tobacco**, another has all the **vanilla** and the other one has all the **matches**. In addition, there is an additional thread named **Agent** who put two ingredients on top of a table. The **agent has endless amounts of all the three ingredients** and he selects randomly the two ingredients to put on top of the table. Once the agent has put the ingredient on the table, then the smoker who holds the complementary ingredient can smoke (but the other two cannot); to do so, it takes the components from the table, make his cigar and smokes it. When he finishes, the cycle starts again:
Agent puts ingredients → Smoker makes a cigar → Smoker smokes → Smorer finishes to smoke → Agent puts ingredients → …
Therefore, at each moment there is, at most, only one smoker smoking.

4. Several processes (**N**) compete for using several resources **Res** with mutual exclusion. A class **Control** supervises the correct usage of such resources. Each process requests a particular amount of resources calling the method **provideRes(id, num)** where **id** is the identifier of the resource and **num** is the amount of resources requested. After the process has used the resources it can release them by calling the method **releaseRes(id, num)**. To assign resources, the Control uses the FCFS technique (First Come, First Use) so processes are served always in the same order in which they send their requests. This way, if a process requires **m** resources and there are some resources already waiting, then it has to wait also even if there are **m** resources available.

   Implement the class **Control** and several threads to simulate requests to it.