1) **RACE CONDITION**

If two processes or more run in parallel and access shared data and they try to change it and then it is possible that there occur some errors, because you don’t know the thread scheduling algorithm and it can be always different. For example the first one can be faster and write in an array at index 4 a number, but the second one can overwrite the array at index 4, before the first one make index++;.

2) **Disabling Interrupts**

I) Mutual exclusion does not work on a multi core machine, nothing will change if you disable one, because the processes are always running on another core at the same time.

II) If Interrupts are disabled, the user process can disable the computer, and it will freeze, because to stop processes is also an interrupt is needed.

3) **Peterson's Solution**

**I1) Process 0 can enter the critical region and do it’s stuff without problems, but if process 1 enters he has to wait till process 0 is finished, cause he has to stay in the while as long as the process 0 is interested and he is looser, after process 0 called “leave\_region” process 1 can enter.**

**I2) Someone is always faster, because there only can exist one looser, and then the same happen like in scenario one.**

**II) The problem here is, that if you want to go in the critical region with one process two times in a row you can’t and you get stuck in the loop until the second process goes in the loop.**

**III)This one is very important, if 2 process enter nearly at the same time, without this variable, both would enter the critical zone.**

**IIII) I would make the interested array bigger, to have place for 3 processes, and I would remove the loser and put there and algorithm which gives the new coming process a number, sorted of course because I want to let them wait like in a queue.**