**Brief Report:** Submit a report showing the critical section of the code ( and logical errors) and

explain your solution in detail.

Race condition occurs when two or more entities tries to access or modify resources in the same system. When events occur in a sequential, the way they are supposed to, then the system would behave correctly. But the problem starts when there is delay that messes up the sequence operation. Then the system will enter a state which was not designed for this system thus failing. Race occurs because sometime this type of failure which requires entities to access the resources at certain order. In software, the race condition happens some events are not properly designed to be able to control its timing. Also to avoid race condition we can design a sequence in which the shared resources are access. So, if one of the even gains the access, we will wait until it finishes its process, or wait for the resources to be available and then give access to the resources to another. Any other events will be automatically blocked when the resources are not busy. These events will be placed in a waiting list in another wait. Once the event that is using the resource is done, then the system will give access to the resource to the first even in the waiting list. Codes protected in this way is called critical solution. Mutual exclusion is the process of granting access to only one process at a time. Having large critical section will definitely affect the performances of the system. In that case refactoring into smaller critical sections can be considered. Another solution is to turn of interrupts when entering or turn on when leaving a critical section.