Shambhoolal Nawaria
21114095

Question
The reader-writer problem is a classic synchronization problem in computer science, where multiple process execution (or thread) competes for shared resources. The reader-writer problem has two types of threads: readers and writers. Readers only read the shared resource, while writers read and write to it. The problem is to ensure that no writer thread is allowed to enter the critical section while a reader thread is accessing the shared resource and vice versa. Additionally, multiple readers can read the shared resources, but only single writers can access shared resources simultaneously.

Answer
implementation of the starvation-free reader-writer problem. Using four variables, two integer types and two Boolean types.

Variable and their use in the program:
1. **Readers**: An integer variable initially set to *0* shows the number of readers in the critical section and ensures that writers wait when readers are in the critical section.
2. **writing**: A Boolean variable initially set to *false* shows that the writer accesses the critical section, and other readers and writers should wait for the critical section.
3. **readersTurn**: A Boolean variable initially set to *false* shows the reader's turn to read the critical section.
4. **waitWriter**: An integer variable initially set to *0* shows the waiting time for writers to access the critical section. When writers wait, then first waiting, writers access the critical section.

How this solution solves starvation problems?

We consider the scenario that when readers first go for critical, which first check *writing || waitWriters && !reaadersTurn* condition initially is the whole condition is false then go to for critical condition and read then when writers come and try to get the critical section it is checked *readers>0 || writing* condition this is true then update *waitWriters* by adding one and wait for critical section. Another reader tries to access the critical section, which checks the *writing || waitWriters && !readers turn* condition, which is true, then wait for the critical section. Here first readers finish his work and come out from the critical section, which decreases *readers* by one, *readerTurn* to false, and if readers==0, then calls informAll() function. In this case, if the second reader tries to access the critical section, then it is checked *writing || waitWriters && !reaadersTurn* condition, which is true due to *waitWriters* variable then wait then writers go for a critical section then checked *readers>0 || writing* condition which is false then access the critical section. Writers are not in starvation. This is the way readers are also not in starvation conditions.