

**FORMAL METHODS BCS2213**

**REPORT**

**SECOND READER WRITER PROBLEM**

**LECTURER NAME: SIR VITALIY MEZHUYEV**

**SECTION: 01G**

**GROUP MEMBERS:**

**1. AHMED SALIH HADI CB12099  
2.FARAH HIDAYAH BT MUSA CB12025  
3.SITI NURBAIZURA BINTI SUKRI CB12042**

**4.NUR NADHIRAH BINTI ABD RASUL CB12023**

**INTRODUCTION**

The readers-writers problem requires the synchronization of concurrent processes simultaneously accessing a shared resource, such as a database object. This problem is different from the known mutual exclusion problem in that it distinguishes between two categories of processes: those who only read the resource, called readers, and those who write it, called writers. Since reader processes only read the resource, it is more efficient to grant all such reader processes simultaneous access to the resource. However, a writer process is granted exclusive access to the resource. The readers-writers requirements allow more concurrency and more efficient use of the resource.

**PROBLEM STATEMENT**

In computer science, the second readers-writers problems are examples of a common computing problem in concurrency. The two problems deal with situations in which many threads must access the same shared memory at one time, some reading and some writing, with the natural constraint that no process may access the share for reading or writing while another process is in the act of writing to it. (In particular, it is allowed for two or more readers to access the share at the same time.) A readers-writer lock is a data structure that solves one or more of the readers-writers problems

**SOLUTION**

EXTENDS Naturals

\\* Naturals are for declare symbols uses.e.g = element (\in)

VARIABLE T1,T2,Mutex

CONSTANT LOCK,UNLOCK, READ, WRITE

TypeInvariant == /\ T1 \in {READ,WRITE}

/\ T2 \in {READ,WRITE}

/\ Mutex \in {LOCK,UNLOCK}

====================================================================

Read == /\ T1 # READ

/\ T2 # READ

/\ Mutex = UNLOCK

Read1 == /\ IF T1 = READ /\ T2 = READ THEN Mutex = UNLOCK

Read2 == /\ IF (T1 = READ /\ T2 = WRITE ) \/ (T2 = READ /\ T1 = WRITE) THEN Mutex = LOCK

Read3 == /\ IF T1 = WRITE /\ T2 = READ THEN Mutex = UNLOCK T1 /\ LOCK T2

\/ ELSE IF T2 = WRITE /\ T1 = READ THEN Mutex = UNLOCK T2 /\ Mutex = LOCK T1

Next == /\ Read1 \/ Read2 \/ Read3

Write1 == /\ IF T1=UNLOCK THEN T2' # UNLOCK ELSE UNCHANGED <<T2>>

====================================================================

THEOREM Write1 => Next /\ []TypeInvariant

====================================================================

**CONCLUSION**

It may be more efficient to allow more than one reader to proceed with simultaneous reading. However, it is not clear to us how this could be achieved without indefinitely locking writers out. We are currently investigating if it is possible to optimize the algorithm to behave more efficiently in such a situation. Furthermore, we would like to consider the complexity implications of our algorithm.