Report

Reader-Writer problem:

- In a problem where, A shared memory among number of concurrent processes
- Some(Readers) wants to read and some(Writers) wants to update the shared memory
- If a write and some other thread(either a reader or a writer) wants to access the shared memory simultaneously, it results in chaos.
- To overcome this, we use 2 semaphores:
    1. Mutex, a semaphore(Initialized to 1), Which is used to ensure **mutual exclusion** when Numreader value is updated. “Numreader” is in the critical section.
    2. Wrt, a semaphore common to both readers and writers
    3. Numreader, Keeps track of how many processes are reading the data.

Reader-Writer Structure:

Writer:

```
While(true)
{
        wait(wrt);
        …
        /*write is performed*/
        /*leaves Critical Section*/
        …
        signal(wrt);
}
```

Reader:

```
While(true)
{
        wait(mutex);
        Numreader++;
        if(readcount==1)
        wait(wrt);
        signal(mutex);
        …
        /*read is performed*/
        …
        wait(mutex);
        Numreader- -;
        if(Numreader==0)
```

```
        /*means no reader in CS*/
        signal(wrt);
        signal(mutex);
        /*reader leaves*/
}
```

Implementation Explanation:

- By Implementing the above solution, we can overcome the Mutual Exclusion: When a reader is executing in Critical section, then writer is waiting till the reader completes its process(Vice-versa), without signal released no other reader or writer thread can enter into critical section
- Implemented program(osa3.c) doesn't lead to starvation or race condition at any circumstances
- Also, The problem in readers-writer problem is priority, There is fairness in the implemented program to avoid starvation. There is no priority for reader or writer, It treats any writer or reader as a regular thread and allows to complete its action