

**Operating System Project**

**Reader Writer Problem**

**Group Members: Asad Ullah Khan (K21-4945)**

**Rohail Rashid Khan (K21-3152)**

**Arham Mehmood (K21-3201)**

**Instructor name: Dr Ghufran Ahmed**

**Objective:**

The main thing was to implement a Reader Writer problem which was further used to synchronize the given object data. Specific number of reader and writer can access the data given under certain limitations through which we have accomplished the given task.

* Readers do not block other readers: Multiple readers should be able to access the resource concurrently, as long as there are no writers currently accessing it. This maximizes parallelism and efficiency by allowing simultaneous read operations.
* Writers have exclusive access: When a writer wants to modify the resource, it should have exclusive access to prevent data corruption or inconsistencies. This means that while a writer is accessing the resource, no other readers or writers should be allowed.
* Fairness and avoiding starvation: The synchronization mechanism should be designed to ensure fairness, so that both readers and writers have a fair chance of accessing the resource. Starvation, where a thread is continuously denied access, should be avoided.

***Introduction:***

In concurrent programming, the reader-writer problem is a classical synchronization problem that arises when multiple threads or processes attempt to access a shared resource.

* The problem revolves around the challenge of maintaining data integrity and ensuring efficient access in scenarios where there are both readers and writers operating on the shared resource.
* The reader-writer problem reflects real-world scenarios where data needs to be accessed and modified by multiple users or processes.
* Consider a situation where a database or a file is shared among several users or a cache is shared among multiple threads in a program. In such cases, multiple readers may require simultaneous access to retrieve information, while writers need exclusive access to modify or update the shared data.
* Implementing an appropriate synchronization mechanism, such as locks, semaphores, or condition variables, can help coordinate the access to the shared resource and enforce the desired behavior.
* Ensuring correct and efficient access in such situations is essential for maintaining data consistency and avoiding conflicts.

***Background:***

Given that the Reader Writer problem deals with the interrelated multiple threads and processes that share a common resource which could be a file. According to this, only two types of resources can have access to the process, reader and writer.

* Reader: who only read the process and don’t modify it.
* Writer: who can read and modify the resource according to the situation.

But this all modification can be achieved with respect to some restrictions.

* Readers and writers cannot access the resource simultaneously. If a writer is currently writing to the resource, no readers should be allowed to read it, and vice versa.
* Multiple readers can access the resource simultaneously without any conflicts.
* Writers should be given priority over readers. If a writer is waiting to access the resource, no new readers should be allowed until the writer has finished.

The choice of solution depends on the requirements and characteristics of the system in which the problem is being addressed.

***Platform:***

The Reader Writer problem is implemented through the below illustrated system,

1. Operating System:

* Linux (Ubuntu 16.04)

1. Locks:

* Semaphore Locks

1. Threads:

* POSIX threads

***Languages:***

The language used in this project is solely (C).

***Methodology:***

1. *The problem itself*

* The reader writer problem wants us to understand the certain implications it works under and identify the shared resource/process that needs to be accessed.

1. *The Mechanism*

* To solve the problem, basic synchronization functions were used which were semaphores, threads, semaphores locks.

1. *The Performance*

* The execution of code in this project was done in a specific way where the reader can have access the critical section of code concurrently but the writer needs to have some special access to the critical section of code given the circumstances.
* The writer in this problem, can access the critical section when the process is not held up by any reader resource. It can change and modify the code and then afterwards release the lock for the reader resource afterwards.
* The reader in this problem can hold the process concurrently. While acquiring the critical section, multiple reader first needs to hold the shared lock and then read the process and afterwards, release the lock after their work is completed for the writer resource.

1. *Testing*

* The final implementation after running through our code through different scenarios including multiple readers, multiple writers and if only reader and writer access the process at the same time.

***Results:***

Below are the attached snap shot of our project.

