Report on Assignment 02
CS 5323 – Design and Implementation of Operating System II
Submitted by – S M Rafiuddin
CWID: A20387294

The synchronization logic in my code uses semaphores to regulate the access to the shared resource by multiple readers and writers. The code uses two semaphores - "mutex" and "wrt". The "mutex" semaphore ensures mutual exclusion between readers, while the "wrt" semaphore ensures mutual exclusion between writers.

When a reader thread wants to access the shared resource, it first waits on the "mutex" semaphore. If the semaphore is available, the thread increments the value of "rcount" and enters the critical section. After completing the critical section, the reader releases the "mutex" semaphore to allow other readers to access the shared resource.

When a writer thread wants to access the shared resource, it first waits on the "wrt" semaphore. If the semaphore is available, the thread enters the critical section by setting the "in_cs" flag to 1. After completing the critical section, the writer releases the "wrt" semaphore and sets the "in_cs" flag to 0.

The choice between mutexes and semaphores depends on the specific synchronization problem being addressed. In general, mutexes are used to provide mutual exclusion between threads accessing a shared resource, while semaphores are used to synchronize access to a shared resource between multiple threads.

In this particular case, semaphores are a better choice because they allow multiple readers to access the shared resource simultaneously while ensuring that only one writer can access the resource at a time. Mutexes, on the other hand, would allow only one thread to access the resource at a time, which could result in unnecessary blocking of other threads. Additionally, semaphores are more flexible than mutexes as they can be used to synchronize access to multiple resources.

Screenshot of the code running in CSX server-

```
srafiud@csx3:~$ gcc A02.c -lpthread
srafiud@csx3:~$ ./a.out 4
Creating Thread 1
Creating Thread 2
Creating WRITER thread
READER 1 thread has started its execution
Critical Section has started
Creating Thread 3
Creating Thread 4
WRITER thread has started its execution.
Exiting from main loop
WRITER thread has completed its execution.
Critical section has ended
READER 1 thread has completed its execution
READER 2 thread has started its execution
Critical Section has started
Critical section has ended
READER 2 thread has completed its execution
READER 3 thread has started its execution
Critical Section has started
Critical section has ended
READER 3 thread has completed its execution
READER 4 thread has started its execution
Critical Section has started
Critical section has ended
READER 4 thread has completed its execution
srafiud@csx3:~$
```