OPERATING SYSTEMS 2
PROGRAMMING ASSIGNMENT 3
Solutions to Readers-Writers and Fair Readers-Writers problems using Semaphores

Implementation :

I implemented the Readers-Writers problem and Fair Readers-Writers problem using Semaphores as discussed in the class in C++.
The Readers-Writers problem states that the critical section may have as many readers as possible but it can have a writer only if there is no other reader or any other writer.

I created nw writer threads followed by nr reader threads. I used rw semaphore as a block for writer threads which takes value 1 and only allows one writer in CS. The code waits on rw when either writer thread tries to enter the CS or there is only one reader in the CS .
When the reader count falls to zero the semaphore is signalled and the writer thread is allowed to progress. Another semaphore is used is the mutex , it is used to change the value of read_count and check if it is zero to allow the writer to progress. If the reader threads keep on coming up then the writer thread will starve forever.

Fair Reader-Writer :

Problem:
starvation of the Writer: a Writer thread does not have a chance to execute while any number of Readers continuously entering and leaving
the working area.

Solution:
Writer indicates to Readers its necessity to access the working area. At the same time no new Readers can start working. Every Reader leaving the working area checks if there is a Writer waiting and the last leaving Reader signals Writer that it is safe to proceed now. Upon completing access to the working area Writer signals waiting Readers that it finished allowing them to access the working area again.

Design Of Program :

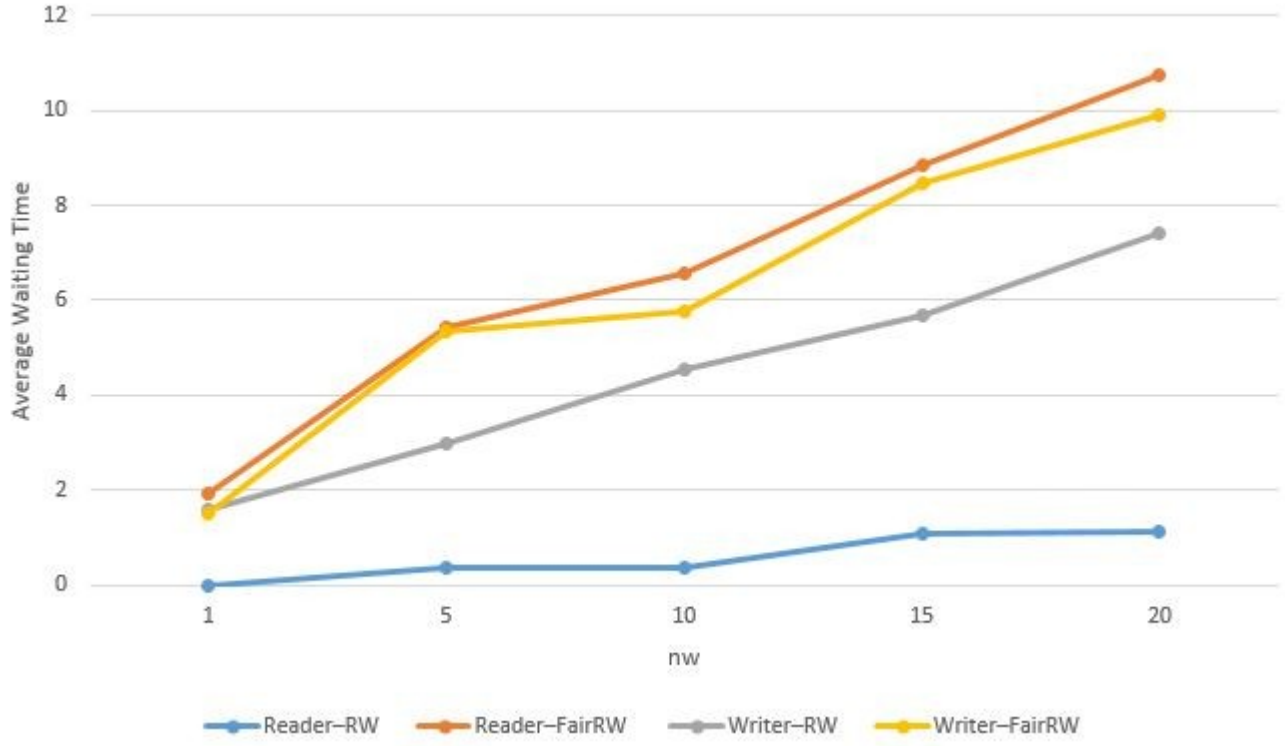
I used two different strategies to solve this problem.
In first, the readers are given absolute priority over the writers and the other one ensures fair chance to both of them.

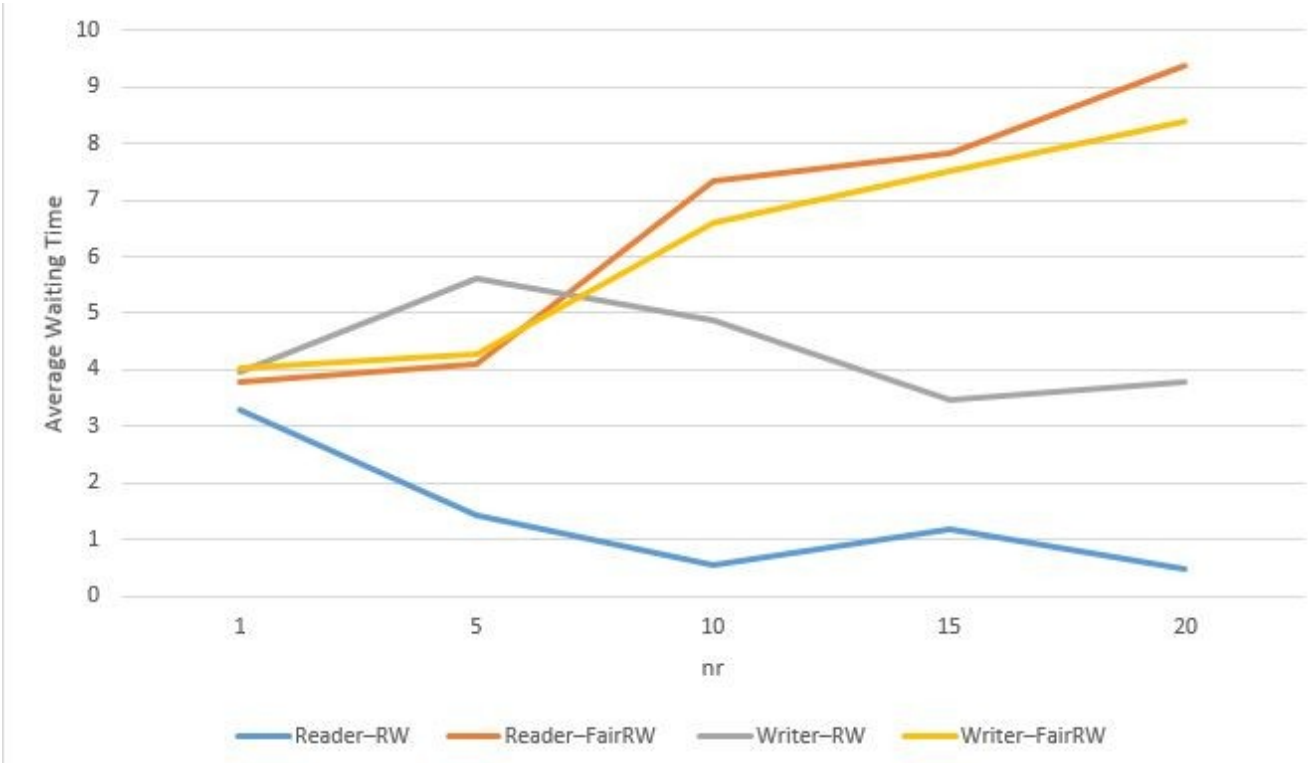
Semaphores : A semaphore is a class of objects which is used to control access to a common resource by multiple processes in a concurrent system. It is similar to a lock but it allows n number of processes to enter the critical section , which is called as the semaphore value. Semaphore has two functions which are atomic in nature which means that only one thread can execute in a semaphore function at a time.

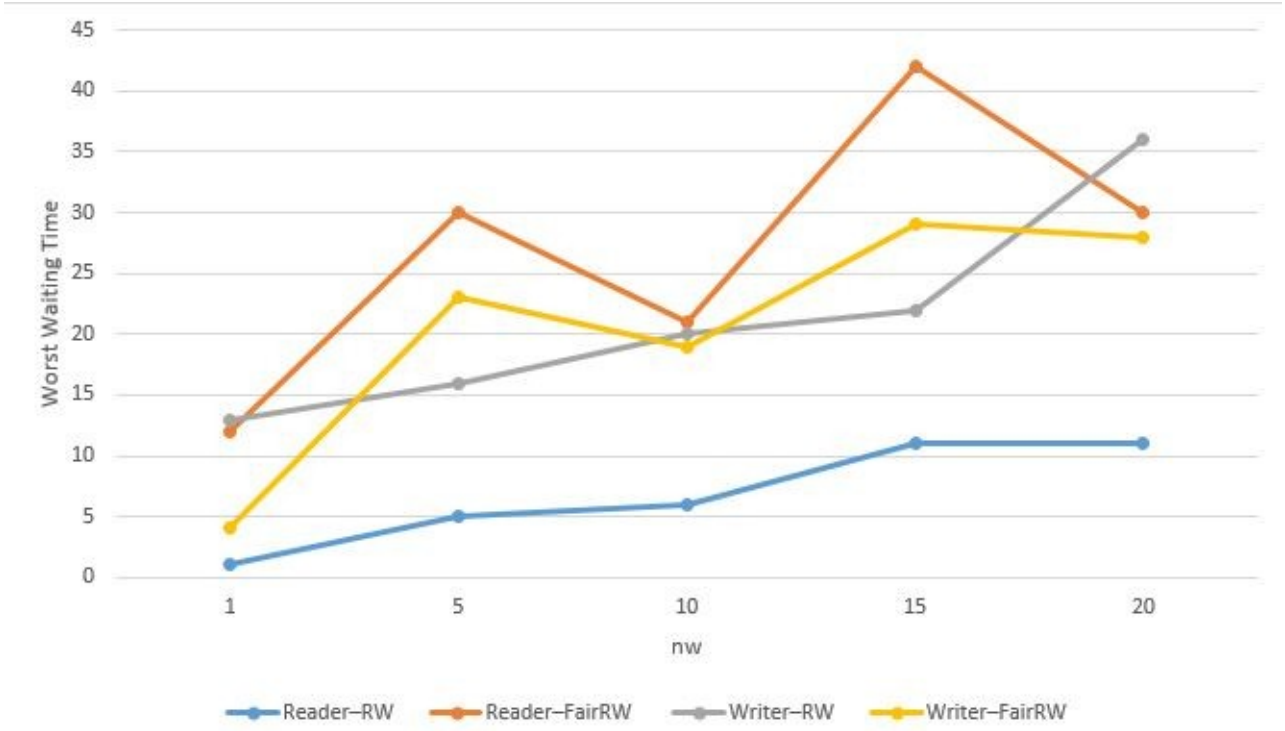
Libraries used :

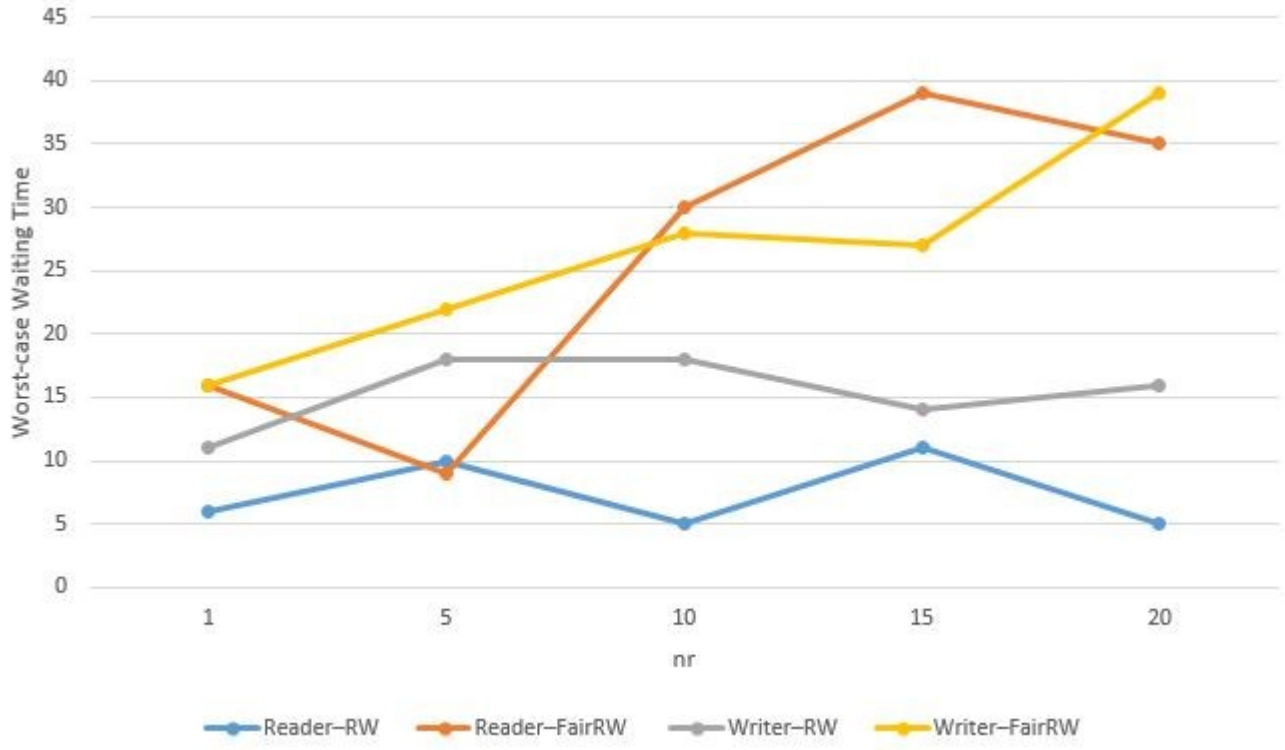
<**thread**> - for thread creation
<**semaphore.h**> - for creating semaphores
<**random**> - To create random number generating engines.
<**unistd.h**> - To calculate current system time.
<**fstream**> - Creating file streams.
<**mutex**> - for creating locks

Graph:


12
10
8
6
4
2
0
Average Waiting Time
1
5
10
15
20
nw
Reader–RW
Reader–FairRW
Writer–RW
Writer–FairRW


10
9
8
7
6
5
4
3
2
1
0
Average Waiting Time
1
5
10
15
20
nr
Reader–RW
Reader–FairRW
Writer–RW
Writer–FairRW


45
40
35
30
25
20
15
10
5
0
Worst Waiting Time
1
5
10
15
20
nw
Reader–RW
Reader–FairRW
Writer–RW
Writer–FairRW


45
40
35
30
25
20
15
10
5
0
Worst-case Waiting Time
1
5
10
15
20
nr
Reader–RW
Reader–FairRW
Writer–RW
Writer–FairRW