SUDHANSHU CHAWHAN
CS16BTECH11037

REPORT
PROGRAMMING ASSIGNMENT 3

Language : C++
Operating system : Ubuntu 16.04

Goal:

The goal of this assignment is to implement a multithreaded solution for the above mentioned problems using Semaphores. You have to implement these two algorithms and compare the time taken for each thread to access the critical section (shared resources).
Implement the algorithms in C++.

The Readers-Writers Problem :-

In the readers-writers problem, there is a shared memory that both reader and writer can access. Reader only reads from the memory while writer can both read and write to the memory. More than one reader can read at the same time. A writer cannot access the memory while a reader is reading. No other thread can access the memory while a writer is accessing the memory.

Design of the program

In the main function, nw writer pthreads and nr reader pthreads are created. Each reader and writer thread is assigned reader and writer functions respectively. Also, all the semaphores are initialised.

Semaphores :-

mutex	-	binary: initialised to 1
rwmutex	-	binary: initialised to 1
avgmutex	-	binary: initialised to 1

Global variable :-

read_count	-	number of readers inside critical section: initialised to 0

Writer function :-
Before entering wait(rwmutex) i.e waits for rwmutex to grant access to the shared variable. Once given access, enter the critical section.
At exit, signal(rwmutex) i.e signals rwmutex to allow other threads waiting for access.

Intuition : writer does not allow any other thread to enter CS therefore a standard wait and signal operation covering access can solve the purpose.

Reader function :-
Before entering, increment read count and checks if read_count is 1, if so wait(rwmutex). Put this inside mutex's wait and signal.
At exit, decrement read count and if read count is zero, signal(rwmutex). Put this inside mutex's wait and signal.

Intuition : reader only allows other readers to enter so check if this is the first reader, if so then stop other writers from entering. If not first reader then just enter the CS. At exit, check if this is the last reader to exit only then allow writers to enter.

The Fair Readers-Writers Problem :-

In the readers-writers problem, there is a shared memory that both reader and writer can access. Reader only reads from the memory while writer can both read and write to the memory. More than one reader can read at the same time. A writer cannot access the memory while a reader is reading. No other thread can access the memory while a writer is accessing the memory. When a writer requests permission, no new reader can read the memory.

Design of the program
In the main function, nw writer pthreads and nr reader pthreads are created. Each reader and writer thread is assigned reader and writer functions respectively. Also, all the semaphores are initialised.

Semaphores :-

in_mutex	-	binary: initialised to 1
out_mutex	-	binary: initialised to 1
wrt_mutex	-	binary: initialised to 0
avgmutex	-	binary: initialised to 1

Global variable :-

in_count	-	number of readers entering critical section: initialised to 0
out_count	-	number of readers exiting critical section: initialised to 0
wait	-	if a writer is waiting to enter: initialised to false

Writer function :-
Before entering wait(in_mutex) then wait(out_mutex). signal(rwmutex) i.e signals rwmutex to allow other threads waiting for access. Check if in_count == out_count then signal(out_mutex) else make wait true and signal(out_mutex) then wait(wrt_mutex) and wait is again false.
At exit, signal(in_mutex)

Intuition : before entering writer stops new threads from entering. If there are no more readers left to read from memory, signal out_mutex else make wait true so that readers know that writer wants to enter. After that, signal out mutex so that other readers can grant writer permission. Now wait for permission. Once permission is given make wait false again.
At exit simply allow other threads to enter.

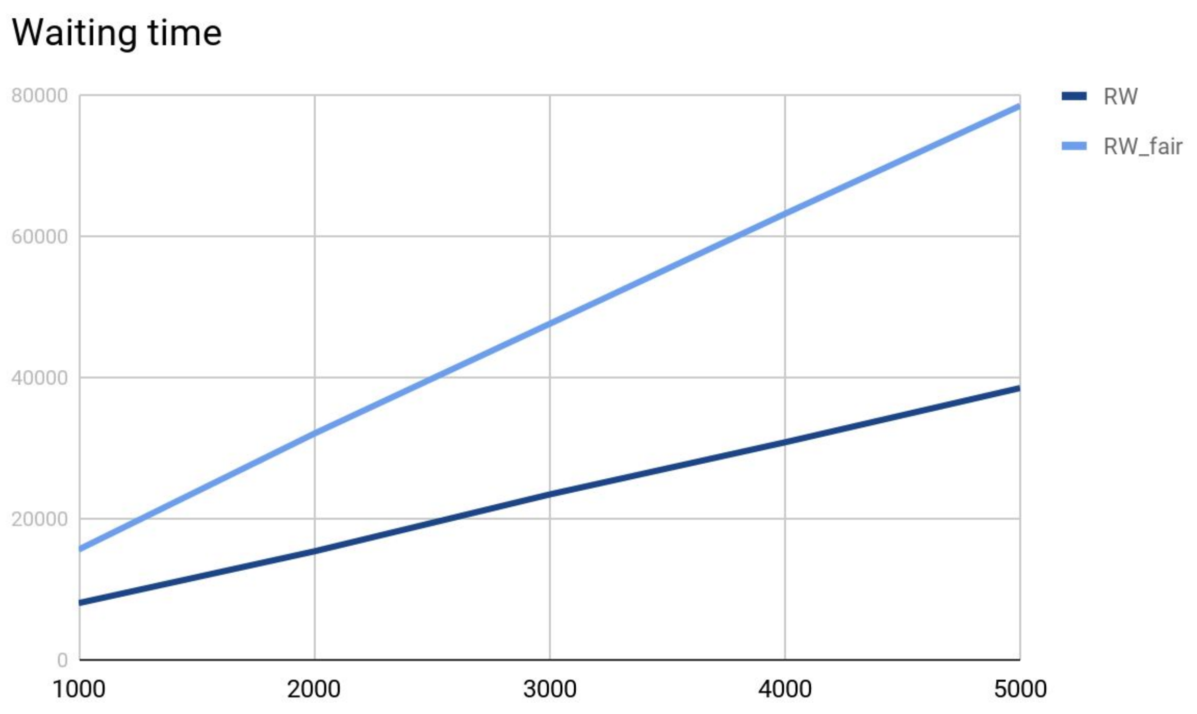
Reader function :-
Before entering, increment in_count and put this inside in_mutex's wait and signal.
At exit, increment out_count. If wait is true and in_count is equal to out_count then signal(wrt_mutex). Put this inside out_mutex's wait and signal.

Intuition : reader first increments the in_count because it's about to enter the CS. after exit, the last reader to exit CS simply checks if a writer is waiting and if it is then allow writer to enter CS.

<u>Complications faced while programming:</u>

- The writer thread was entering before reader thread could exit. After some debug, this issue was solved.

Graphical analysis of the algorithms:-



We can see that the graph is a straight line. This means that waiting time is proportional to k.

Also, waiting time of RW_fair is always greater than RW and RW_fair’s waiting time increases more faster than RW’s as we increase k.