Operating Systems 2
SPRING SEMESTER 2018
Programming Assignment 3
Solution to Reader-Writer using Semaphores

INSTRUCTOR: DR. SATHYA PERI
Mayank Hooda -cs16btech11022@iith.ac.in

0. Task :

Our task it to implement readers writers synchronization using semaphores. The 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.

1. Design of the Program :

We use two different strategies to solve the reader-writer problem. In one of them *Readers* are given absolute priority above writers and the other one is fair in its sorts.

Critical Section :

Ad segment of code which we want only a single thread to access at a given time is called a critical section.

Entry Section :

A part of code which is available to all the threads where the threads contend to enter into the Critical Section. **Bounded wait** guarantees that no thread waits forever in the entry section and gets a fair chance to enter Critical section.

Exit Section :

A segment of code wherein a thread exits from the critical section and opens the lock for other threads to enter the CS.

Waiting Time :

The time that a thread spends in the entry section waiting to get into the critical section in called waiting time.

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. The functions are :

Wait(*sem_t value***)**:

The wait method blocks a given process if *value<=0* . At every call , the wait method decrements *value* by 1 and henceforth allows **n** number of processes to pass through depending on *value*

Post(*sem_t value***)**:

The postl method signals the **wait** method to release a blocked process, if blocked and increments the *value* by 1 and hence if *value* again crosses 0 then more number of processes will be able to go through.

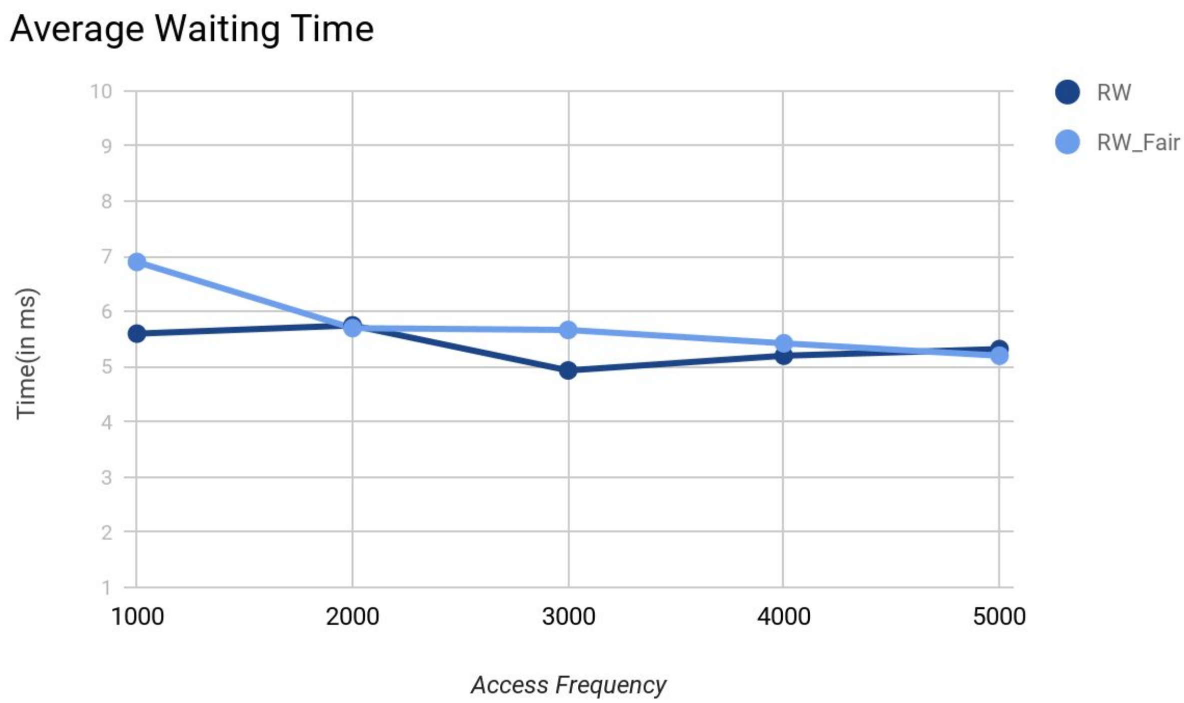
2. Implementation of the Program :

- We make **nr** reader and **nw** writer threads using **pthread_create** function and give them the *writer* and *reader* function to operate on respectively.
- k*r[ ]* and *kw[ ]* are two arrays which store the count of frequencies every thread has to go through.

- We use **rw** semaphore as a block for *writer* threads which takes the initial value as 1 and thus allows only one writer in CS. We wait on rw when either a writer thread tries to enter the CS or there is only 1 *reader* thread in the CS. When the reader count falls down to zero then we *post* on **rw** semaphore and allow the writer thread to progress.
- Another semaphore that we use is **mutex**.Mutex is used in the reader thread to update the **read_count** and check if it is zero or not to allow the writer thread to proceed.
- The problem with this approach is that writers will starve if readers keep on coming and never fall down to zero.
- So to mitigate this problem , we introduce another semaphore **in** which is initialised to 1. This blocks both the reader and writer thread. Thus when a *writer* thread is blocked and a *reader* thread comes along then it will be blocked until a *writer* thread is free. Thus here writers don't starve and neither do readers.

Important Libraries Used :

- **<pthread.h> :** Used for creation and work assignment of the threads.
- **<semaphore.h> :** Used for implementing the semaphore , provides three crucial functions *sem_init* , *sem_wait* , *sem_post*.
- **<random> :** Used for the random number generator for seeding the sleep interval.
- **<unistd.h> :** Used for calculating the current system time.



Graph Plotted for following parameters :

nw = 5 nr = 5

1000<=kw = kr <=5000

CSseed = 1

RemSeed = 2

The graph shows that the according to waiting times , the **efficiency** of the algorithms is :

RW > RW_Fair

But this is purely in terms of time , if we consider the distribution of waiting time among processes , it is more uniform than RW wherein the writers are starving and having a much higher waiting time.