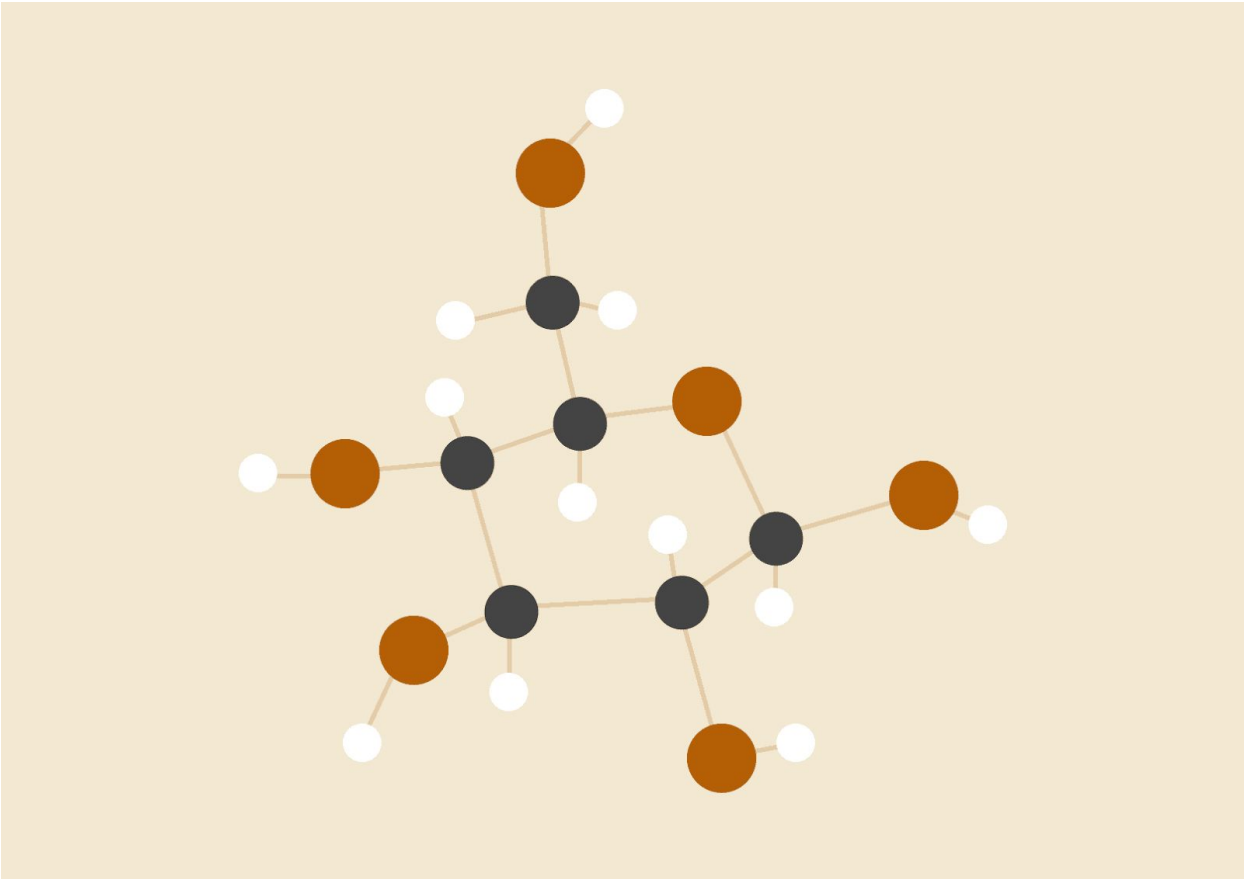
OS ASSIGNMENT 4

Solving reader-writer problem using semaphore



Happy Mahto

19-03-2019
CS17BTECH11018
DR. Sathya peri

INTRODUCTION

This assignment solves the Reader - Writer problem using semaphore, in two ways. One presented in OS book, and one in research paper published by Vlad Popov and Oleg Mazonka.

HYPOTHESIS

Critical Section : Segment of code which is supposed to be accessed by only one thread at a time.

Entry Section : Segment of code which is available to all the threads and where threads contend to enter in critical section. Bounded waits guarantees that no threads waits for forever in critical section and gets a fair chance to enter in critical section.

Exit Section : Segment of code wherein thread exits critical and opens the lock for other process to enter in critical section.

Waiting Time : time that a threads spends in entry section waiting to enter in critical section.

Semaphore : an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait() and signal().

IMPLEMENTATION

In Solution provided for Reader Writer problem in book, Writer threads would starve if a continuous stream of reader threads keep coming and and reading from the shared file/data.

Fair Reader - Writer solution uses an extra semaphore to solve this problem. This is implemented such as -- reader thread coming after writer thread should wait for writer thread to complete its job.

- Programs uses input file “inp-params.txt” to accept input values. It accepts *capacity, no of producer threads, no of consumer threads, producer frequency , consumer frequency, time for producer , time for consumer threads.*
- Program uses default_random_generator generator to generate random number

with exponential_distribution, they are contained in distribution1, and distribution2.

- Main thread creates n_w and n_r writer thread and reader threads respectively. Each reader thread executes the Reader function K_r times. Each Writer thread executes the function K_w times.
- **Semaphore implementation**
    1. Semaphore are used from c++ <semaphore.h> library.
    2. Semaphore are defined and initialized as :

        Sem_t <name of the semaphore>

        sem_init(&<name of the semaphore >, { 0, 1} , value of semaphore)

        "The second argument in sem_init can be 1 or 0, 1 if semaphores are shared among process and 0 if semaphore are shared among threads.

- In normal Reader writer solution, the program uses two semaphore "mutex" and "rw_mutex" both initialized to 1.

  Whenever reader thread comes it first acquires semaphore "mutex" and checks if the read_count is 1 it acquires semaphore "rw_mutex" so no writer thread can writer when reader threads are reading.

  Reader threads performs reading in critical section, two reader can exist in critical section at same time, but no writer thread is writing at the same time when a reader thread is reading.

  After reading is performed it checks if the read_count is 0 then it releases semaphore "rw_mutex" so that writer thread can write, and releases mutex lock.

- In Fair Reader writer solution, the program uses three semaphore "mutex", "rw_mutx" and "in" each initialized to 1.

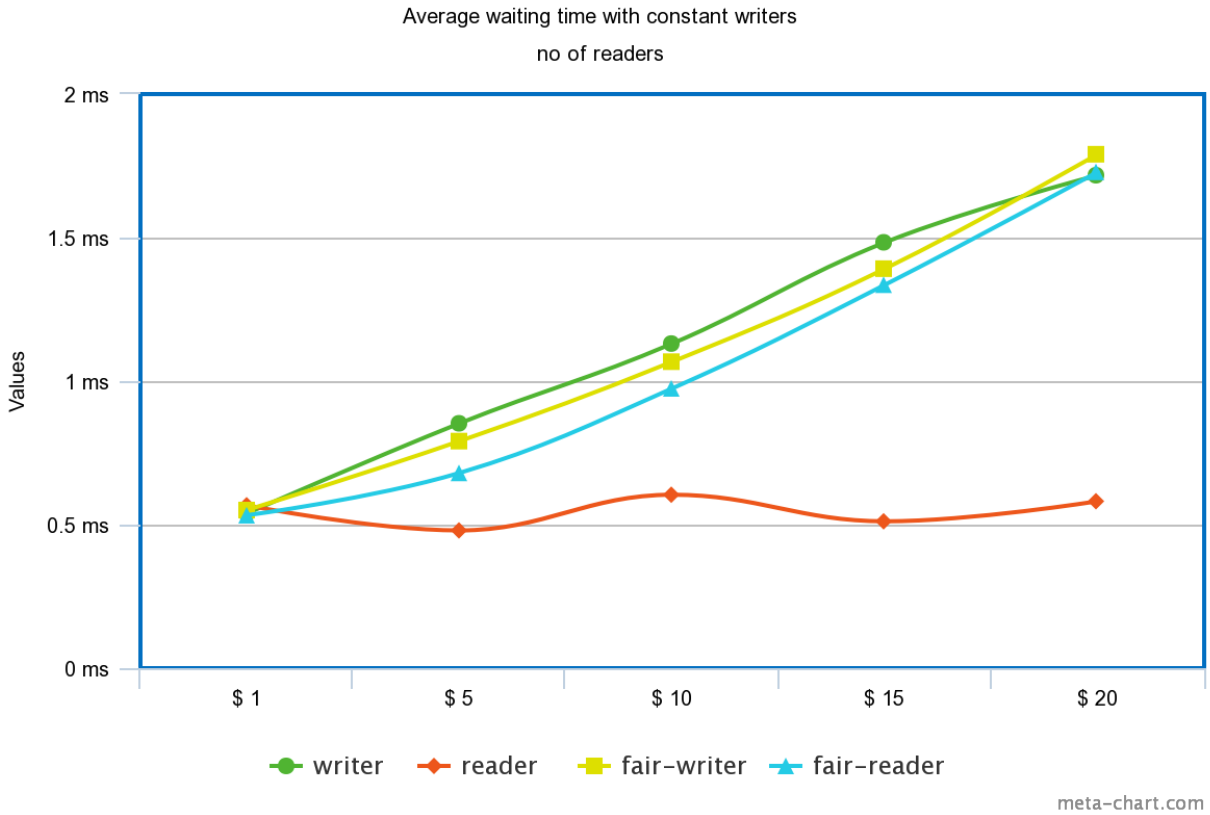
  Whenever a reader thread comes it acquire the semaphore "in" and then process is same as normal reader writer solution. But reader thread are made to wait for writer thread as it has previously occupied semaphore "in", and all the reader threads coming after this writer thread should wait till the writing is done.

GRAPH

1. Average waiting time with constant writers.

   The Graph contains four curves -

   - Average time taken by the reader threads to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().
   - Average time taken by the writer threads to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().

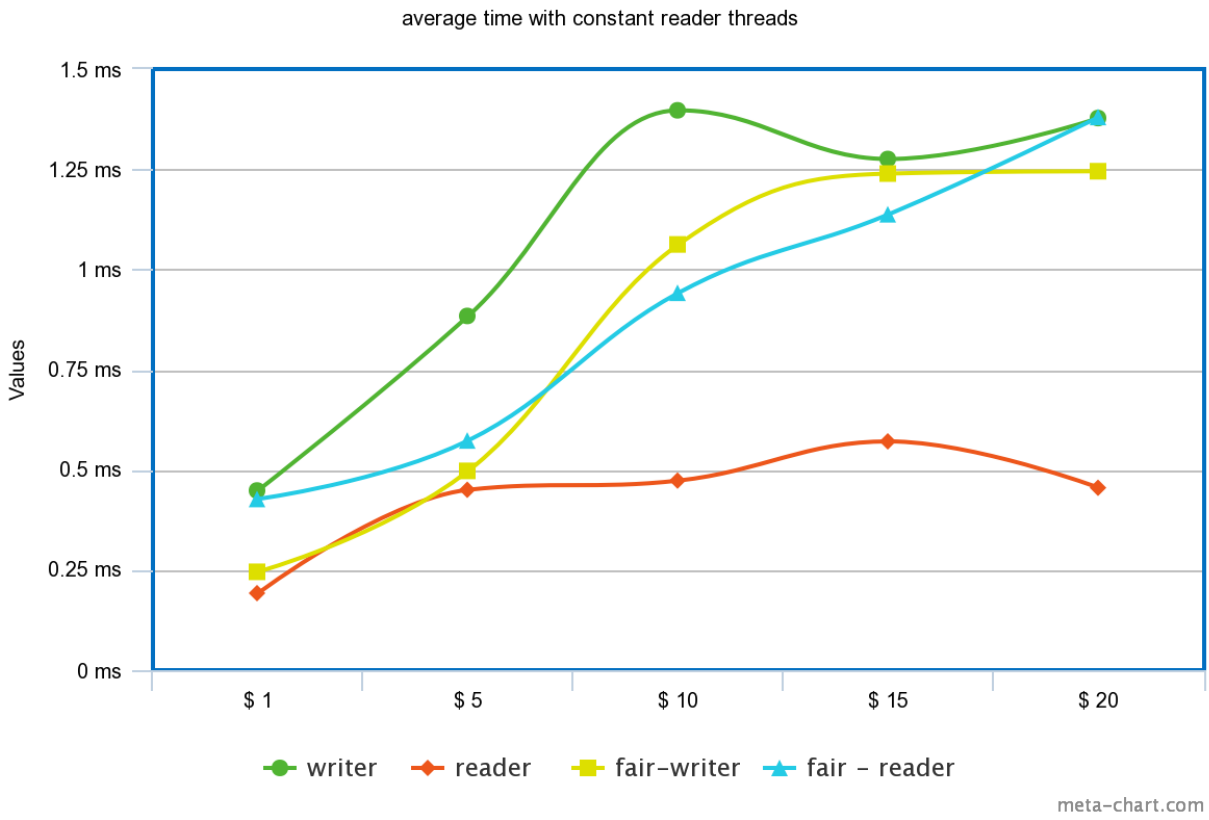


2. Average waiting time with constant readers

   The Graph contains four curves -

   - Average time taken by the reader threads to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().

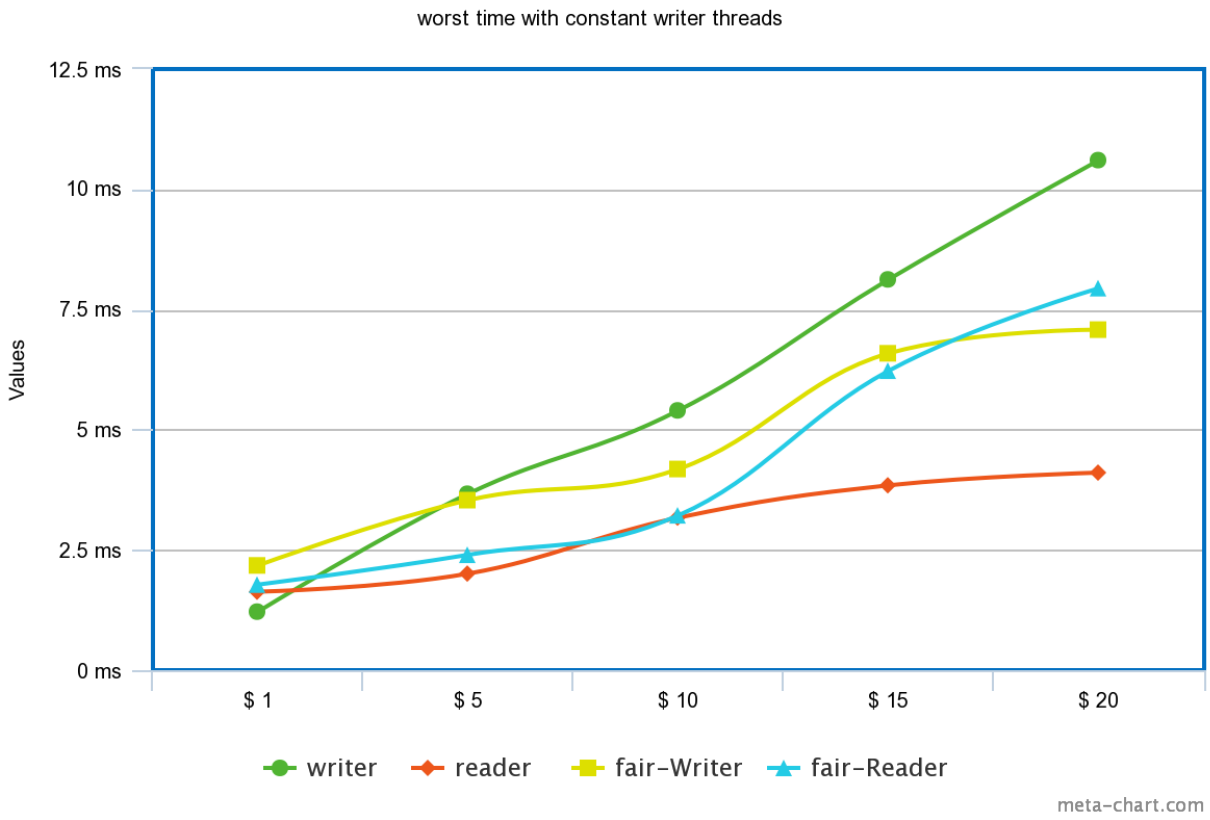
- Average time taken by the writer threads to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().



3. Worst time with constant writer threads

Graph contains four curves :

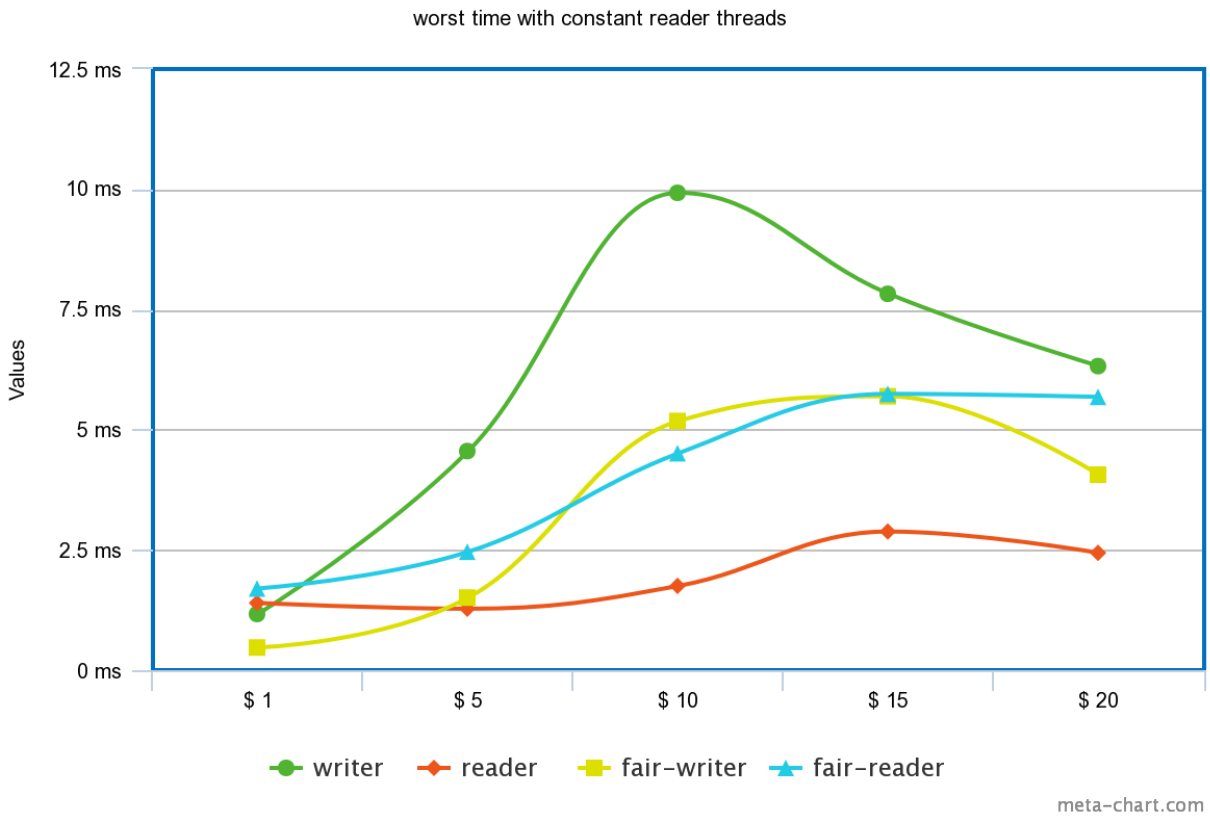
- Worst-case time taken by the reader threads to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().
- Worst-case time time taken by the writer threads to enter the CS for each algorithm: Readers Writers() and Fair Readers Writers().



4. Worst time with constant reader threads

Graph contains four curves :

- Worst-case time time taken by the writer threads to enter the CS for each algorithm : Readers Writers() and Fair Readers Writers().
- Worst-case time time taken by the reader threads to enter the CS for each algorithm:Readers Writers() and Fair Readers Writers().



CONCLUSION

- The average waiting for writer threads is more than average time waiting for reader, but in fair reader writer solution the gap between average time of reader and writer is less than the gap in normal reader writer solution. This is because in fair reader writer solution a reader whom request after a reader cannot get into critical section. This reader thread must wait for writer thread to complete its execution.
- The worst time for writer threads is more than the worst time for reader threads, because writer threads should wait for all the reader threads to complete their execution.