OS2 CS3523

Name: Swapnil Bag

Roll no: ES22BTECH11034

Aim: To solve the Readers-Writers problem (writer preference) and Fair Readers-Writers problem using Semaphores and thread library of C++

Low level design

- Firstly the input (nr , nw, kr, kw, randRemTime, randCSTime) are accepted using a command line argument
- A structure _th_param_ is created to store thread parameters like id , kr, kw, randRemTime, randCSTime
- Reader and writer threads are created in the main function and the respective parameters are passed to the reader and writer threads

- **Writer Preference solution**

  1) Count of readers and writers is initialized to 0. 4 semaphores r_lock(reader lock), w_lock(writer lock), r_try(reader trylock), resource (resource lock) and a print_l (to synchronize printing to file).
  2) In the reader function, each thread makers kr requests to read. First it enters and tries to acquire the r_try lock and the r_lock (so that other readers do not enter simultaneously). The count of readers is incremented and if there is a single reader the resources is locked ,it releases the r_lock and the r_try lock enters the critical section where it sleeps for a random time and then exits the critical section by invoking r_lock and if there are no more readers it signals the writers.
  3) The writer threads makes kw requests to read, it acquires the w_lock, increments count of writers, if there is a single writer you lock the r_try semaphore so that no reader can enter, release w_lock and acquire resource. In the critical section after finishing it releases resource. While exiting it updates the count of writers and releases the r_try
  4) The main purpose of the r_try lock is that whenever a reader wants to access the CS, each reader should individually acquire the r_try and release it, but in case of writers if a single writer acquires it then an incoming stream of writers can be serviced (each writer need not individually acquire it). This ensures writers preference but readers can be starved

- **Fair Reader writers solution**

  1) Count of readers and writers is initialized to 0. 4 semaphores r_lock(reader lock), resource (resource lock) and a print_l (to synchronize printing to file) ad a serv_q lock that in a way ensures fairness among the threads
  2) In the reader function, each thread makers kr requests to read. First it enters and tries to acquire the serv_q lock and the r_lock (so that other readers do not enter simultaneously). The count of readers is incremented and if there is a single reader the resources is locked ,it releases the r_lock and the serv_q, lock enters the critical section where it sleeps for a random time and then exits the critical section by invoking r_lock and if there are no more readers it signals the writers.
  3) The writer threads makes kw requests to read, it acquires the serv_q, and then acquires the resource so that no other thread can enter. While exiting it releases the resource lock.

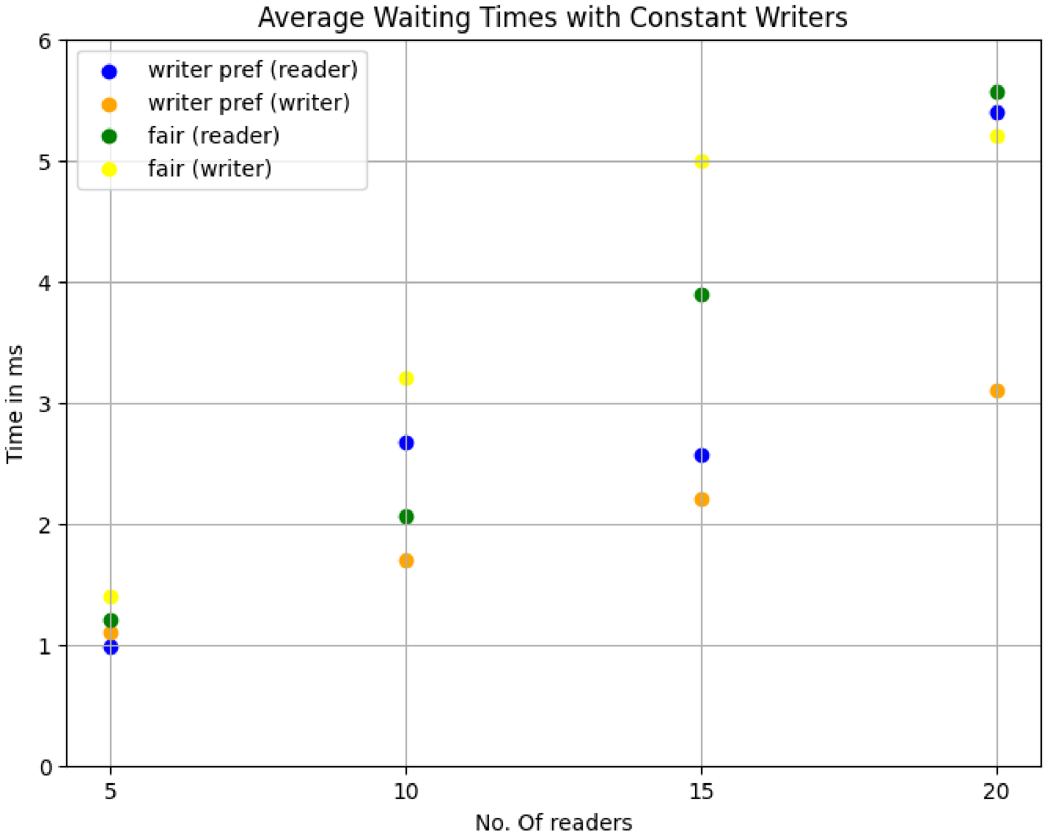
4) The main purpose of the serv_q lock is to ensure fairness among the reader and writer threads. In a way each reader or writer must invoke this and release it individually, so there’s an equal chance of both the threads to acquire which in turn ensures fairness.

- **Output** : For each solution The logs is printed in log.txt and Average_time.txt. In the log file the time stamp is provided in hh:mm:ss:microsecond format for better precision, and In the average time file, the average and worst case time from request to entry is printed

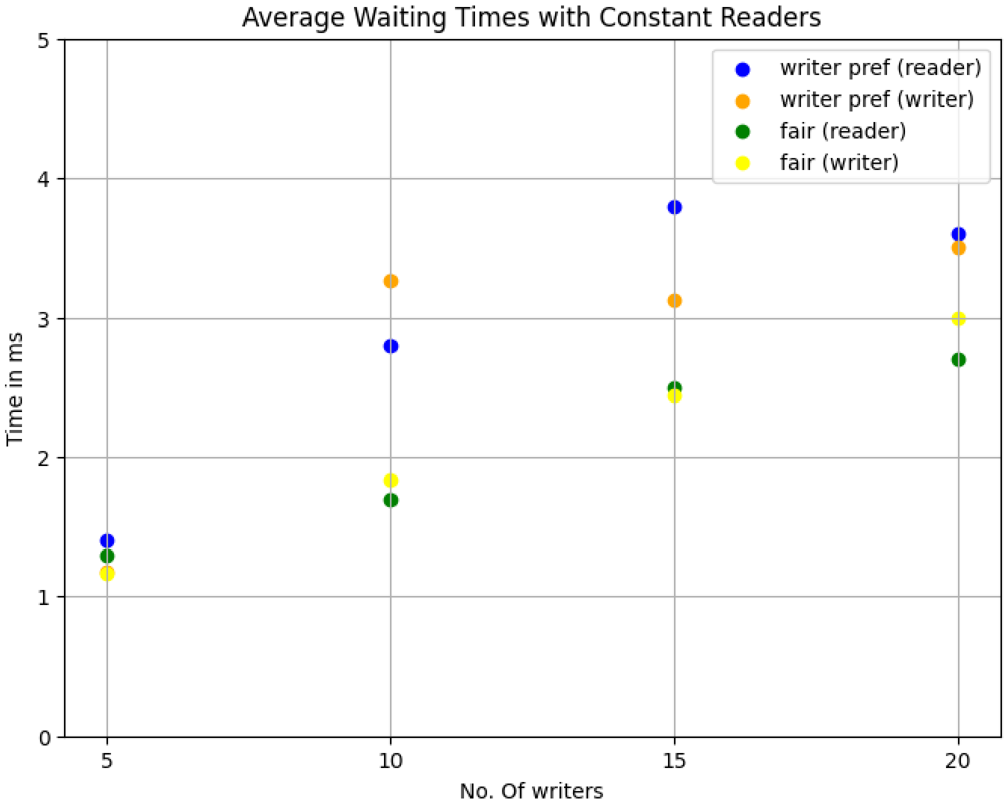
Analysis:

We can see that in the writer preference code the average waiting time for reader threads is usually greater than the average waiting time of writer thread , which holds roughly in the worst case also. This worst case time is mainly contributed due to multiple lock contentions. In the log file we can observe that initially readers request for CS however over time the writer threads execute in one chuck as it acquired the lock for themselves, thus resulting in a preference for writers. When almost all writers finish the readers complete

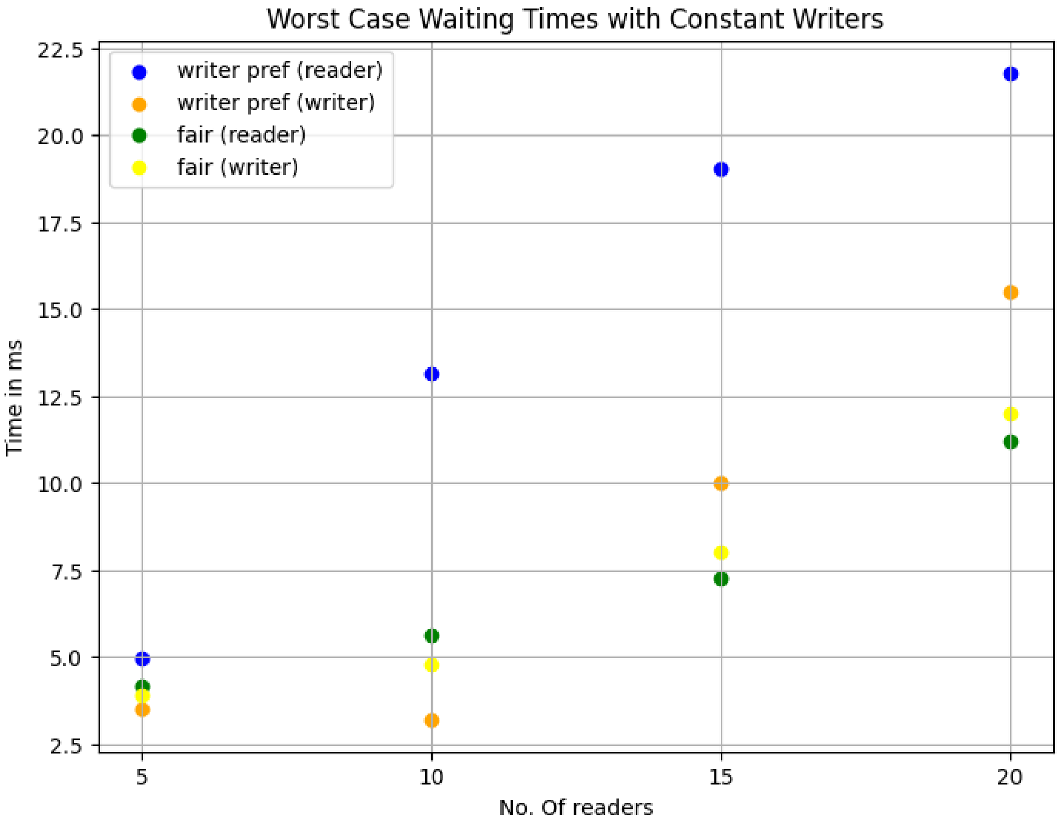
In the fair code The average as well as the worst case time is nearly equal for both type of threads. This is possibly because of the simple lock usage and lock acquiring sequence for both type of threads. Furthermore from the log file we can infer that reader and writer execution is evenly interspersed indicating both type of threads get a fair chance to execute

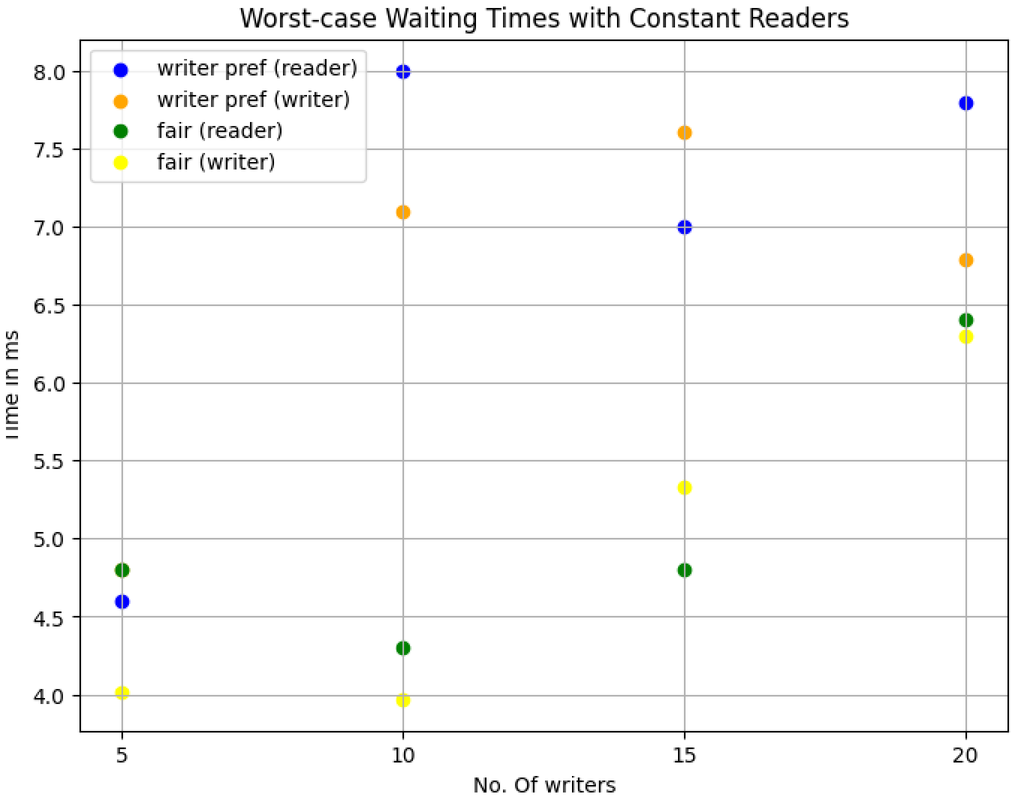


Average Waiting Times with Constant Writers
writer pref (reader)
writer pref (writer)
fair (reader)
fair (writer)
Time in ms
0
1
2
3
4
5
6
5
10
15
20
No. Of readers



Average Waiting Times with Constant Readers
writer pref (reader)
writer pref (writer)
fair (reader)
fair (writer)
Time in ms
0
1
2
3
4
5
5
10
15
20
No. Of writers


Worst Case Waiting Times with Constant Writers
writer pref (reader)
writer pref (writer)
fair (reader)
fair (writer)
Time in ms
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
5
10
15
20
No. Of readers


Worst-case Waiting Times with Constant Readers
writer pref (reader)
writer pref (writer)
fair (reader)
fair (writer)
Time in ms
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
5
10
15
20
No. Of writers